ScreenShot

COLLANA DI PROGRAMMAZIONE

COME PROGRAMMARE PASSO PER PASSO

COMMODORE 64

LIBRO 2

Tutte le tecniche per realizzare fantastici giochi e la grafica più spettacolare - interamente a colori -

PHIL CORNES

Screen Shot

COLLANA DI PROGRAMMAZIONE

COME PROGRAMMARE PASSO PER PASSO

COMMODORE 64

LA COLLANA DI PROGRAMMAZIONE SCREEN SHOT

Non c'è mai stato, come oggi, un bisogno così urgente di una serie di guide pratiche, facili, ben fatte, per imparare a usare il computer. La collana Screen Shot è stata concepita proprio per questo. È un concetto completamente nuovo nel campo dell'autoistruzione per i calcolatori. Ed è la prima collana di manuali dedicati a macchine specifiche, completamente illustrati, che insegnano a programmare passo per passo.

LIBRI SUL COMMODORE 64

Questo è il secondo volume di una serie di guide, uniche nella loro concezione, che insegnano a programmare passo per passo il Commodore 64. Insieme con gli altri volumi, questo libro costituisce un corso di programmazione completo e autosufficiente, che inizia dai principi di base e prosegue, descrivendo programmi e tecniche via via sempre più sofisticati, fino a un livello avanzato.

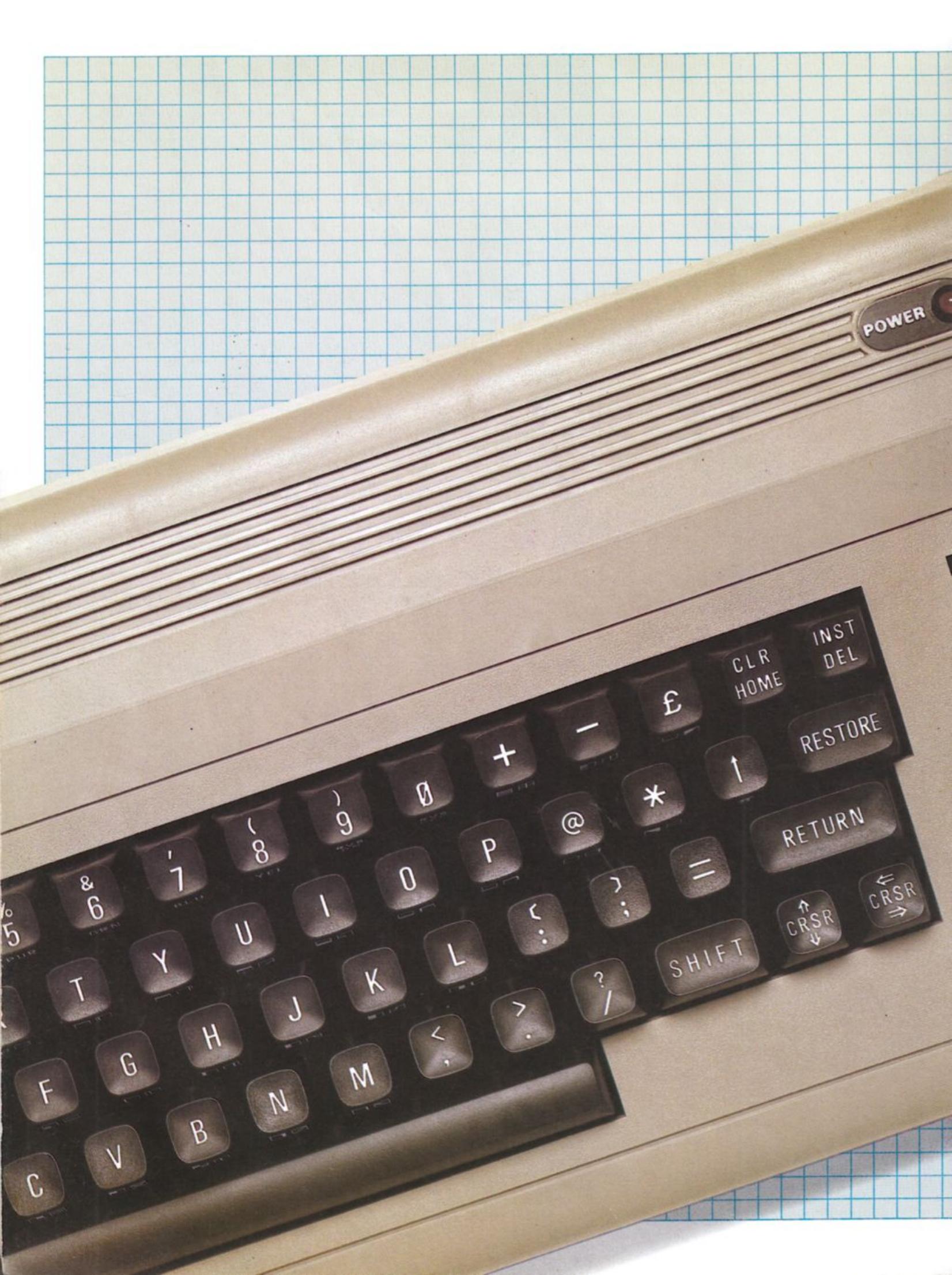
NELLA STESSA COLLANA

Come programmare passo per passo lo
ZX Spectrum e ZX Spectrum plus

PHIL CORNES

Dopo aver studiato matematica e programmazione, Phil Cornes ha lavorato alla realizzazione di sistemi educativi basati su computer al National Training College del British Telecom. Dal 1978 lavora anche come autore di pubblicazioni tecniche e collabora a varie riviste sui personal computer, come *Personal Computer World*, *Computing Today* e *Electronics Today International*. Ha scritto un libro e molti articoli sull'uso e sulla programmazione del Commodore 64.

SUPERNOVA

LIBRO 2

ScreenShot
COLLANA DI PROGRAMMAZIONE
COME PROGRAMMARE PASSO PER PASSO
COMMODORE 64
PHIL CORNES
SUPERNOVA
LIBRO 2
f 1
f 3
f 5
f 7

SOMMARIO

La collana Screen-Shot
è stata ideata e disegnata
da Dorling Kindersley Limited,
9 Henrietta Street, Covent Garden,
London WC2E 8PS.

Designer Roger Priddy
Fotografie Vincent Oliver
Editor David Burnie
Art Editor Peter Luff
Managing Editor Alan Buckingham

Traduzione
di Paolo Ferrara
e Marco Mezzalama
Edizione italiana a cura
di Marco Maiocchi
Step-by-step programming
for the Commodore 64
Book two

Via Matteo Bandello, 8
20123 Milano

ISBN 88-377-0001-6
62a-I-85

Finito di stampare
nel marzo 1985
da Officine Grafiche A. Mondadori
Verona
Printed in Italy

CODICE
COSTO
N.
NETTO
TASSA
LORDO
PROVA CON UN'ALTRA ALIQUOTA? 12.5
28
26
24
22
20
18
16
14
12
1 2 3 4 5 6 7 8 9 101112

COME DEFINIRE E USARE LE FUNZIONI

Tutti i computer forniscono un certo numero di funzioni predefinite, cioè comandi che consentono di trasformare un numero in un altro secondo una specifica legge. Le funzioni producono un risultato che potrà essere utilizzato più tardi nello svolgimento del programma; SQR (SQuare Root = radice quadrata) e INT (INTeger = parte intera) sono esempi di funzioni predefinite per il Commodore: questi comandi, partendo da un numero dato, operano su di esso producendo un altro numero. L'insieme di funzioni predefinite per il Commodore è ampio, ma se volete usare funzioni che non sono presenti nel BASIC Commodore, non dovete riscriverne le istruzioni relative ogni volta: potete programmare il vostro computer così da ottenere specifiche sequenze di calcoli.
Queste sequenze, o funzioni, sono richiamate per mezzo del comando FN (FunctioN) e sono definite dal comando DEF FN (DEFine FunctioN).

Come scrivere le funzioni per i vostri programmi

Per usare una funzione dovete innanzitutto definire cosa essa dovrà fare, per mezzo di una istruzione di definizione.
Per esempio l'istruzione:

```
120 DEF FNA(X)=4*X+36
```

definisce una funzione chiamata "A". Il numero su cui una funzione opera è conosciuto come il suo argomento: in questo caso l'argomento è X. La funzione prende qualunque valore di X le venga fornito, lo moltiplica per 4 e quindi gli somma 36; se in un programma volete assegnare il valore 10 all'argomento di questa funzione, dovete farlo usando la parola chiave FN nel seguente modo:

```
200 PRINT FNA(10)
```

Questa istruzione visualizzerà il valore assunto dalla funzione quando X è sostituito dal valore 10, cioè 4*10+36 ovvero 76.
Una volta che una funzione è stata definita in un programma, potete usare essa stessa e il suo argomento come una qualsiasi variabile o costante numerica del programma medesimo: per esempio voi potete addizionare, sottrarre, moltiplicare o dividere le funzioni e i loro argomenti e, persino, utilizzare una data funzione come argomento di un'altra funzione. A meno che voi non stiate affrontando problemi di analisi matematica, è molto improbabile che sfruttiate intensamente queste istruzioni, ma per problemi più semplici, le funzioni sono facili da utilizzare e utili nel rendere i programmi più leggibili.

Cosa possono fare le funzioni

Il seguente programma mostra un semplice modo in cui potete usare delle funzioni per produrre un risultato numerico che verrà poi stampato. Esso converte la distanza di una stella, misurata in anni luce, nella corrispondente distanza misurata in miglia.
La funzione che realmente effettua la conversione è definita alla linea 50: essa moltiplica il valore dato per 5.88:

PROGRAMMA DISTANZE STELLARI

```
LIST
10 PRINT CHR$(147) : POKE 53280,0 : POKE
53281,0
20 POKE 214,5 : PRINT
30 PRINT TAB(6);"PROGRAMMA DISTANZE STEL
LARI"
40 PRINT TAB(6);"▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬
▬▬▬"
50 DEF FNC(L)=L*9.461
60 PRINT : PRINT : PRINT
70 PRINT TAB(2);"DISTANZA DELLA STELLA I
N ANNI LUCE": PRINT : PRINT TAB(15);
80 INPUT L
90 PRINT :PRINT : PRINT
100 PRINT "LA STELLA E";FNC(L);"MIGLIAIA
 DI MILIARDI"
110 PRINT : PRINT TAB(5);"DI MIGLIA DIST
ANTE DALLA TERRA"
READY.
```

Trattare i problemi relativi all'uso di una funzione in questo caso, può sembrare inutile, ed è infatti improbabile che voi userete l'istruzione FN in un programma semplice. Provate però ad immaginare che cosa potrebbe succedere se voi voleste effettuare il calcolo un certo numero di volte in punti differenti dello stesso programma, e con differenti valori; è in questo caso che le funzioni che voi stessi potete definire dimostrano tutta la loro potenza: quando l'espressione della funzione è lunga e complicata, definirla solamente una volta vi permette di creare le linee di programma che effettuano i calcoli in modo molto più semplice da scrivere e da verificare. L'istruzione FN è simile a una subroutine, costituita da un unico comando, che opera solo su valori numerici. Dal momento che un'espressione contenente FN rappresenta realmente un numero, potete usarla per sostituire qualsiasi tipo di calcolo comunque complesso. Quando scrivete le vostre funzioni, in effetti, state fornendo al computer dei comandi che il suo programma residente, il BASIC, non possiede, estendendo così le possibilità del linguaggio stesso.

Come usare le funzioni in una sequenza di calcoli
Supponiamo che vogliate calcolare il costo di qualcosa venduto a metri quadrati (potrebbe trattarsi di moquette). Dovete moltiplicare la lunghezza e la larghezza di ogni stanza per calcolare la superficie del pavimento e, quindi, moltiplicare questa superficie per il costo della moquette per unità di area: se chiamate la lunghezza e la larghezza rispettivamente con X e Y, e il costo unitario con Z, il costo per stanza sarà dato da:

$$(X*Y)*Z$$

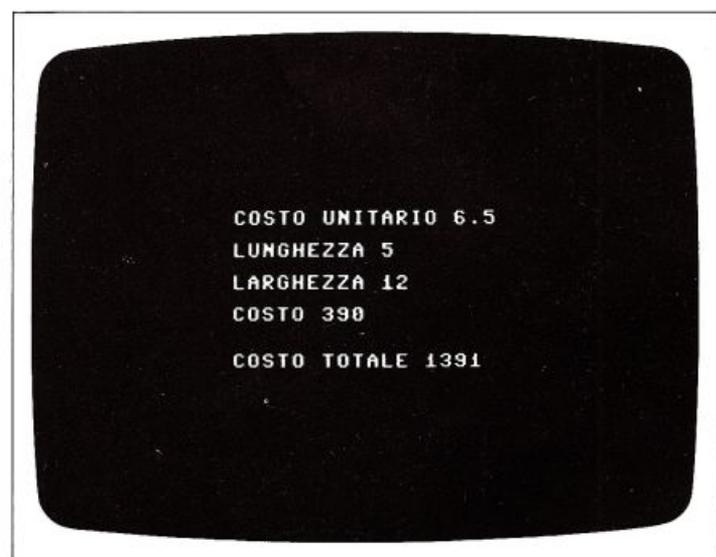
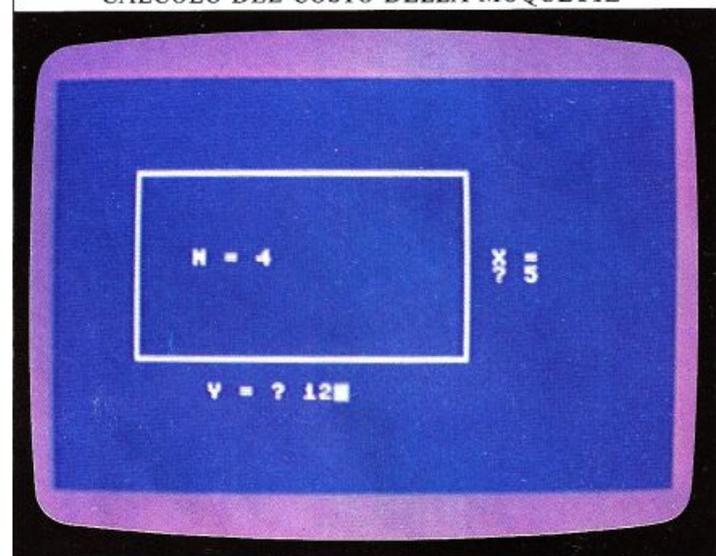
Nel programma seguente il costo di ogni stanza è calcolato da una funzione definita alla linea 10 e chiamata C. Questa funzione è usata proprio alla fine del programma alle linee 340 e 350, dopo che sono stati forniti al programma i valori di X e Y insieme al valore del costo per unità di area, il quale è richiesto alla linea 50:

PROGRAMMA "CALCOLO DEL COSTO DELLA MOQUETTE"

```
10 PRINT CHR$(147) : DEF FNC(Z)=(X*Y)*Z
 : POKE 53280,0 : POKE 53281,0
20 PRINT "PROGRAMMA CALCOLO" : PRINT "DE
L COSTO DELLA MOQUETTE" : PRINT : PRINT
30 INPUT "QUANTE SONO LE STANZE DA CONSI
DERARE";N : PRINT
40 PRINT "QUALE E' IL COSTO UNITARIO"
50 INPUT "DELLA MOQUETTE";P
60 T=0 : FOR C=1 TO N
70 PRINT CHR$(147)
80 POKE 214,4 : PRINT : POKE 211,5 : POK
E53280,4 : POKE 53281,6
90 PRINT CHR$(176);
100 FOR K=1 TO 20
110 PRINT CHR$(99); : NEXT K
120 PRINT CHR$(174)
130 FOR K=1 TO 10
140 POKE 211,5 : PRINT CHR$(98);
150 POKE 211,26 : PRINT CHR$(98)
160 NEXT K
170 POKE 211,5 : PRINT CHR$(173);
180 FOR K=1 TO 20

READY.
```

```
LIST 190-

190 PRINT CHR$(99); : NEXT K
200 PRINT CHR$(189)
210 POKE 214,9 : PRINT : POKE 211,9
220 PRINT "N =";C
230 POKE 214,2 : PRINT : POKE 211,6
240 POKE 214,9 : PRINT : POKE 211,28
250 PRINT "X ="
260 POKE 211,28 : INPUT X
270 POKE 214,17 : PRINT : POKE 211,10
280 INPUT "Y =";Y
290 PRINT CHR$(147) : POKE 53280,0 : POK
E53281,0
300 POKE 214,6 : PRINT : POKE 211,8
310 PRINT "COSTO UNITARIO";P
320 PRINT : PRINT TAB(8);"LUNGHEZZA";X
330 PRINT : PRINT TAB(8);"LARGHEZZA";Y
340 PRINT : PRINT TAB(8);"COSTO";FNC(P)
350 T=T+FNC(P) : PRINT : PRINT
360 PRINT TAB(8); "COSTO TOTALE";T
370 FOR Z=1 TO 4000: NEXT Z : NEXT C
READY.
```

Le linee dalla 70 alla 290 costruiscono un'immagine grafica che riproduce il contorno di una stanza, e quindi attendono che voi immettiate i valori della sua lunghezza e della sua larghezza (potete usare l'unità di misura che preferite, a patto che sia la stessa usata per il costo unitario). Una volta che avete fatto ciò, il programma acquisisce i valori e quindi cancella lo schermo.
La schermata successiva richiede il costo per unità di area (per metro quadro o qualunque altra unità di misura abbiate usato) e quindi usa questo valore per dirvi quanto costerà la moquette necessaria per quella stanza.
Così come è stata utilizzata la funzione C per produrre questo risultato, essa è impiegata nuovamente alla linea 350 per aggiornare il totale provvisorio: il programma esegue queste istruzioni una volta per ogni stanza e, dopo ogni passata, scoprirete che la riga COSTO TOTALE nella seconda schermata sarà aggiornata per mostrarvi il totale provvisorio dei costi calcolati.

"CALCOLO DEL COSTO DELLA MOQUETTE"

Potete definire una funzione in qualsiasi punto di un programma, tenendo presente che la definizione della funzione deve essere immessa prima che la funzione stessa sia richiamata; normalmente, la cosa migliore è inserire le definizioni delle vostre funzioni all'inizio del programma. Nel programma "CALCOLO DEL COSTO DELLA MOQUETTE", l'uso di una funzione rende più semplici le linee 340 e 350; quando si tratta di calcoli molto complessi le funzioni diventeranno essenziali per rendere un programma facile da capire.

COME POTENZIARE LE DECISIONI DEL BASIC

Le parole chiave del BASIC IF e THEN consentono ad un programma di operare in un certo modo fino a quando non si verifichi la condizione specificata nell'istruzione di IF; quando ciò accade, il programma viene forzato a eseguire uno svolgimento diverso. Le potenzialità di IF...THEN però non si esauriscono nella realizzazione di una semplice decisione del tipo "sì" o "no": combinando IF...THEN con le parole chiave AND e OR potete far sì che questo comando sia in grado di gestire situazioni molto più complicate.
Dal momento che la sintassi del BASIC è definita in modo da rispecchiare l'ordine in cui le parole sono usate nel linguaggio naturale, potete usare IF...THEN proprio come se steste descrivendo un insieme di condizioni a qualcuno. Il programma seguente vi mostrerà come potete utilizzare il generatore di decisioni IF... THEN ad un livello più sofisticato.

PROGRAMMA "PALLINA CHE RIMBALZA"

```
LIST -200

10 S=54272 : POKE S+24,15
20 POKE S+5,0 : POKE S+6,240
30 POKE 53280,0 : POKE 53281,0
40 POKE S+4,16 : PRINT CHR$(147)
50 FOR R=3 TO 18
60 POKE 214,R : PRINT CHR$(129)
70 PRINT TAB(11);CHR$(18);" ";
80 PRINT TAB(26);" "
90 POKE 214,3 : PRINT : POKE 211,R+8
100 PRINT CHR$(18);" "
110 POKE 214,18 : PRINT : POKE 211,R+8
120 PRINT CHR$(18);" "
130 NEXT R
140 X1=INT(RND(0)*10)+14 : Y1=15
150 D1=0.9 : D2=-1
160 X2=14 : Y2=INT(RND(1)*10)+6
170 D3=-1 : D4=0.8
180 POKE 214,Y1 : PRINT : POKE 211,X1
190 PRINT " " : POKE S+4,16
200 POKE 214,Y2 : PRINT : POKE 211,X2

READY.
```

```
LIST 210-

210 POKE 214,Y : PRINT : POKE 211,X
220 PRINT CHR$(113)
230 IF X<13 OR X>25 THEN DX=-DX : F=1
240 IF Y<5 OR Y>16 THEN DY=-DY : F=1
250 IF F=0 THEN 180
260 POKE S+4,17
270 GOTO 180
READY.
```

Le linee dalla 40 alla 140 costruiscono un contorno rettangolare arancione al centro dello schermo; le linee 160 e 170 specificano la posizione di partenza e la direzione di una palla (un carattere grafico della tastiera) all'interno del rettangolo. Per simulare il movimento della palla, la linea 200 calcola continuamente le nuove coordinate della stessa; dopo di ciò le linee 230 e 240 controllano se la palla ha raggiunto uno dei lati del rettangolo, cioè se il numero di riga della stessa (coordinata y) è minore del lato superiore del rettangolo o maggiore del lato inferiore, oppure se il numero di colonna (coordinata x) è maggiore del lato destro o inferiore al sinistro. Se una di tali condizioni è verificata, le linee 230 e 240 modificano la direzione del moto orizzontale o verticale della palla, a seconda delle necessità.

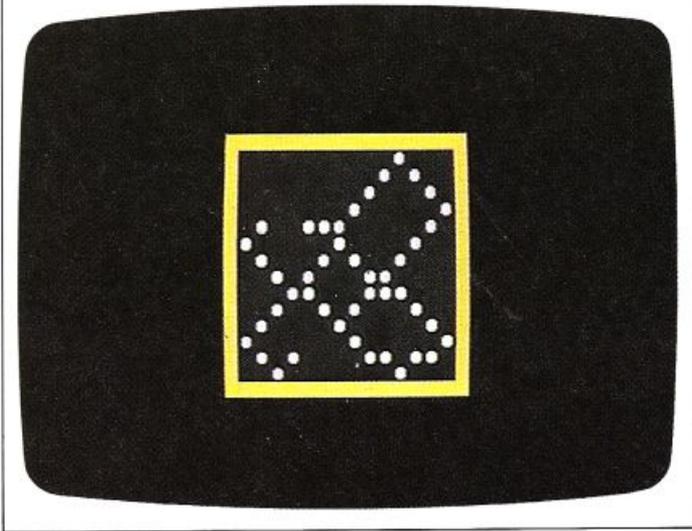
Come legare le decisioni tra loro
Potete ora innalzarvi un gradino più in alto rispetto al programma precedente per imparare come più condizioni possono essere incorporate in un'unica istruzione di IF...THEN. Il programma seguente fa uso del connettivo logico AND per verificare se una serie di condizioni si verificano contemporaneamente o meno:

PROGRAMMA "PALLINE CHE RIMBALZANO"

```
LIST -200

10 S=54272 : POKE S+24,15
20 POKE S+1,115 : POKE S,88
30 POKE S+5,0 : POKE S+6,240
40 POKE 53280,0 : POKE 53281,0
50 PRINT CHR$(147)
60 FOR R=3 TO 18
70 POKE 214,R : PRINT CHR$(129)
80 PRINT TAB(11);CHR$(18);" ";
90 PRINT TAB(26);" "
100 POKE 214,3 : PRINT : POKE 211,R+8
110 PRINT CHR$(18);" "
120 POKE 214,18 : PRINT : POKE 211,R+8
130 PRINT CHR$(18);" "
140 NEXT R
150 PRINT CHR$(159)
160 X=15 : Y=15
170 DX=1.2 : DY=-1
180 POKE 214,Y : PRINT : POKE 211,X
190 POKE S+4,16
200 X=X+DX : Y=Y+DY : F=0

READY.
```

```
LIST 210-

210 PRINT " "
220 X1=X1+D1 : Y1=Y1+D2 : F=0
230 X2=X2+D3 : Y2=Y2+D4
240 POKE 214,Y1 : PRINT : POKE 211,X1
250 PRINT CHR$(156);CHR$(113)
260 POKE 214,Y2 : PRINT : POKE 211,X2
270 PRINT CHR$(159);CHR$(113)
280 IF X1<13 OR X1>25 THEN D1=-D1 : F=1
290 IF Y1<5 OR Y1>16 THEN D2=-D2 : F=1
300 IF X2<13 OR X2>24 THEN D3=-D3 : F=2
310 IF Y2<5 OR Y2>17 THEN D4=-D4 : F=2
320 IF X2<=X1+1 AND X2>=X1-1 AND Y2<=Y1+
1 AND Y2>=Y1-1 THEN F=3
330 IF F=0 THEN 180
340 IF F=1 THEN POKE S+1,115 : POKE S,88
 : GOTO 370
350 IF F=3 THEN POKE S+1,55 : POKE S,80
 : GOTO 370
360 POKE S+1,85 : POKE S,88
370 POKE S+4,17 : IF F=3 THEN 10
380 GOTO 180
READY.
```

Questo programma è molto simile al precedente eccetto che, ora, sono presenti due gruppi di linee che stampano un simbolo grafico che cambia continuamente la propria posizione sullo schermo. Inoltre, la posizione iniziale delle due palline è determinata a caso alle linee 140 e 160, ed esse sono animate per mezzo delle linee numerate da 180 a 380. La schermata seguente mostra cosa succederebbe se voi eliminaste le righe 190 e 210 che cancellano le posizioni precedenti delle palline.

IMMAGINE DELLE PALLINE CHE RIMBALZANO

La seconda pallina è predisposta in modo da partire in una direzione differente da quella della prima e ad una velocità verticale leggermente diversa, cosicché le due palline abbiano una grande probabilità di scontrarsi, altrimenti esse si inseguirebbero all'interno del rettangolo lungo la stessa traiettoria. La linea 320 è quella in cui il computer prende una complessa decisione riguardo la posizione di entrambe le palline: senza questa linea, qualora esse si scontrassero, continuerebbero il proprio movimento come se nulla fosse successo. Ciò non costituirebbe certamente una simulazione convincente di quello che succede in realtà, perciò la linea 320 decide quando le due palline sono sufficientemente vicine per provocare una collisione; tale linea comprende una decisione IF...THEN con tre AND per controllare se x1 e x2 sono abbastanza vicini, e se lo sono anche y1 e y2. Questa operazione viene effettuata considerando x2, per esempio, e decidendo se esso è minore o uguale di x1 +1 e, contemporaneamente, maggiore o uguale di x1-1. Se sono verificate tutte queste condizioni, significa che le due palline occupano la stessa posizione, oppure posizioni adiacenti, nel qual caso si assume che esse debbano collidere: verrà emesso un "beep" sonoro e l'intero processo ricomincerà.

Come usare l'istruzione IF...THEN nei giochi

Come avete appena visto IF...THEN è molto utile se volete conoscere se due caratteri stanno occupando la stessa posizione sullo schermo; questa possibilità è usata frequentemente nei programmi in cui un carattere deve essere "colpito" da un altro:

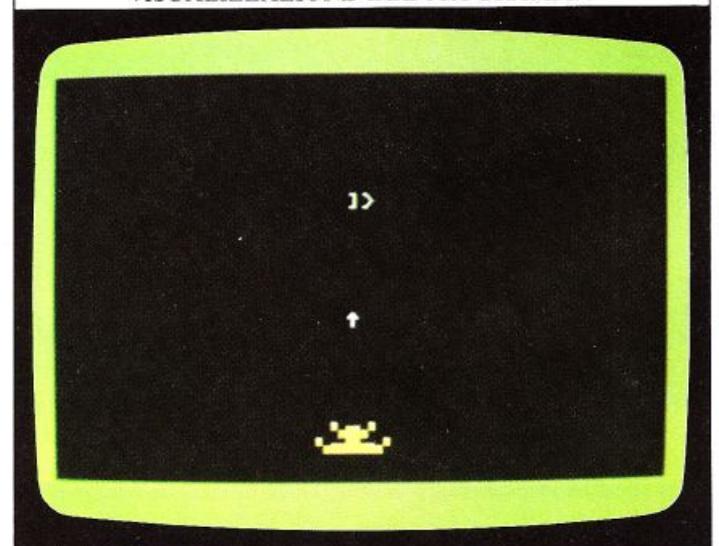
PROGRAMMA "POSTAZIONE DI TIRO"

```
10 PRINT CHR$(147);CHR$(158)
20 POKE 53280,5 : POKE 53281,0
30 POKE 214,20 : PRINT
40 PRINT TAB(18);CHR$(191);CHR$(18);CHR$
(183);CHR$(191)
50 PRINT TAB(17);CHR$(191);CHR$(18);CHR$
(184);" ";CHR$(184);CHR$(191)
60 X=7 : Y=19
70 POKE 214,Y : PRINT
80 PRINT TAB(19);" "
90 POKE 214,6 : PRINT
100 PRINT TAB(X);"  "
110 X=X+1 : Y=Y-1
120 IF X>36 THEN X=5
130 IF Y<1 THEN Y=19
140 POKE 214,6 : PRINT
150 PRINT TAB(X);CHR$(153);"]>"
160 POKE 214,Y : PRINT
170 PRINT TAB(19);CHR$(5);"↑"
180 IF X=18 AND Y=6 OR X=19 AND Y=6 THEN
 200
190 GOTO 70
200 FOR T=1 TO 1000 : NEXT T : GOTO 10
READY.
```

Questo programma disegna una postazione di tiro sulla parte inferiore dello schermo: essa spara delle frecce a un bersaglio orizzontale che vola continuamente attraverso lo schermo. La linea 180 controlla se le coordinate video della freccia sono identiche a quelle del bersaglio e, se lo sono, il programma "salta" indietro alla linea 10 e ricomincia da capo; in caso contrario esso "salta" alla linea 70 e muove tutti i caratteri di una posizione. La riga 130 controlla se la freccia ha raggiunto il limite superiore dello schermo mentre la riga 120 verifica se il bersaglio è arrivato al bordo destro dello stesso. Quando eseguite il programma scoprirete che la freccia sparata dalla postazione di tiro colpisce il bersaglio quando esso si trova ad un quarto circa del suo passaggio sullo schermo: ciò accade perché il programma utilizza delle coordinate fisse; d'altra parte, se inserite la seguente linea

```
60 X=INT(RND(0)*10+1):Y=19
```

il risultato diventerà casuale e cambierà ad ogni riesecuzione del programma stesso.

VISUALIZZAZIONE DEL PROGRAMMA

SCANSIONE DELLA TASTIERA

Per l'immissione di dati in un programma in esecuzione, voi avete fino ad ora usato l'istruzione INPUT; con l'istruzione INPUT i dati immessi da tastiera saranno forniti al programma dopo la pressione del tasto RETURN. Questa tecnica presenta alcuni svantaggi; per esempio, potreste dimenticarvi di premere il tasto RETURN e, comunque, il dover premere sempre in sequenza due tasti rallenta l'esecuzione del programma. È molto più comodo rendere il Commodore in grado di rispondervi senza aspettare la pressione del tasto RETURN, proprio come i video giochi, che reagiscono tutte le volte che viene premuto un tasto. Per fare in modo che il Commodore si comporti così, potete usare l'istruzione GET.

Come il computer riconosce i caratteri

Come avete potuto osservare nel primo volume, tutti i simboli che il Commodore riconosce sono rappresentati come valori numerici tra 0 e 255, secondo il codice ASCII (American Standard Code for Information Interchange); il computer utilizza questo codice tutte le volte che voi immettete un numero o una stringa di caratteri per determinare il suo contenuto.
Troverete l'intero codice ASCII a pagina 61, ma potete chiedere al computer di stampare numeri e lettere con il loro codice immettendo queste istruzioni:

```
10 FOR N=48 TO 90
20 PRINT N; "-"; CHR$(N),
30 NEXT N
```

Apparirà sullo schermo una lista parziale del codice ASCII:

LISTA DEL CODICE ASCII

Il prossimo programma usa i due comandi BASIC GET e ASC per rispondere quando premete un tasto alfabetico, modificandolo e quindi stampando il carattere specificato dal nuovo codice.
Il risultato è un codificatore della tastiera che fornisce un messaggio codificato ad ogni pressione di tasto.

PROGRAMMA "CODIFICATORE DELLA TASTIERA"

In questo programma la riga 20 stampa una intestazione, e quindi la riga 30 usa GET per scandire la tastiera; qualsiasi carattere sia stato immesso mediante la pressione di un tasto durante la scansione della tastiera è contenuto nella variabile A$. Se non premete nessun tasto durante la scansione, alla variabile è assegnato il valore nullo.
La seconda parte della riga 30, controlla se la variabile ha valore nullo, e se è così, il programma ritorna all'inizio della riga 30, cosicché la tastiera risulta essere continuamente scandita.
Quando il computer rileva la pressione di un tasto, il controllo è passato alla linea 40; se il tasto premuto non è una lettera, le righe 40 e 50 passano a loro volta il controllo alla linea 90, la quale visualizza il carattere rilevato. Se il tasto premuto è una lettera, la riga 60 converte questo carattere nel suo codice ASCII, usando l'istruzione ASC (che, quindi, lavora al contrario dell'istruzione CHR$), e assegna questo valore alla variabile A. Le righe 70 e 80 operano su A per trasformare questo valore in un altro e convertirlo in un carattere (A$) che è stampato dalla riga 90.
Il programma sembra visualizzare messaggi senza senso quando battete un messaggio, ma non esiste un codice che non possa essere decodificato, e voi potete facilmente trasformare il programma in un decodificatore cambiando le seguenti tre istruzioni:

```
20 PRINT "DECODIFICATORE DI CARATTERI"
60 A=ASC(A$)-2
70 IF A 65 THEN A=A+26
```

Per vederlo al lavoro, provate a battere i seguenti caratteri quando il decodificatore è in esecuzione:

SWGUVQ G' WP OGUUCIIKQ FK VGUV

Come utilizzare i tasti funzione

Nello stesso modo in cui usate GET per la scansione dei tasti alfabetici o numerici, potete operare per rileva-

re se uno dei tasti funzione è stato premuto, poiché ad ognuno di essi è assegnato un valore del codice ASCII, benché, normalmente, non visualizzino nulla sullo schermo. Il codice ASCII dei tasti funzione va da 133 a 140, come sarà dimostrato dal seguente programma:

PROGRAMMA "RILEVAZIONE DEI TASTI FUNZIONE"

Tutte le volte che premete un tasto funzione, il suo simbolo apparirà sullo schermo; questo non vi sembrerà molto utile, ma in realtà vi mostra come questi tasti sono usati nella programmazione: se usate GET potete far controllare dai tasti funzione delle operazioni predefinite; potete disporre di un totale di otto differenti funzioni che possono andare dalla definizione dei colori, per esempio, all'animazione degli sprite.

Provate i vostri riflessi con GET

Il "jiffy" clock del Commodore, trattato nel primo volume, è un contatore a tre byte che è incrementato 60 volte al secondo per registrare il tempo trascorso da quando il computer è stato acceso; inoltre il Commodore possiede alcuni semplici comandi che vi permettono di accedere al clock e di utilizzare i valori che vi sono memorizzati.

Provate questo comando immediato:

```
PRINT TI
```

Fatto ciò, quello che appare è un numero che indica quanti sessantesimi di secondo sono trascorsi dall'accensione del computer. Ora immettete un'altra istruzione:

```
PRINT TI$
```

Questa volta viene visualizzata la medesima informazione, ma convertita in ore, minuti, secondi, con due cifre ciascuna. Per esempio: 013000 significa che è trascorsa esattamente un'ora e mezza da quando avete cominciato ad usare il computer. Così come siete in grado di leggere il clock in questi due modi, potete anche azzerarlo, o assegnargli qualsiasi valore numerico voi vogliate, per mezzo di questo tipo di istruzione:

```
TI$="000000"
```

Che azzera il clock.

Potete usare GET in combinazione con il clock per misurare la velocità di qualsiasi cosa durante l'esecuzione di un programma. Un modo semplice per usare questa combinazione è un programma che rileva i vostri tempi di reazione. Eccone uno: sceglie a caso una lettera e stabilisce il tempo che impiegate a trovare e premere il tasto corrispondente alla lettera selezionata:

PROGRAMMA "MISURATORE DI RIFLESSI"

```
LIST
10 PRINT CHR$(147) : POKE 53280,0 : POKE
 53281,0
20 POKE 214,5 : PRINT : POKE 211,6
30 PRINT "MISURA I TUOI RIFLESSI CON"
40 PRINT : PRINT TAB(10)
50 PRINT "IL PROVA RIFLESSI"
60 FOR T=1 TO 1500 : NEXT T
70 A$=CHR$(INT(RND(0)*26)+65)
80 POKE 214,10 : PRINT : POKE 211,8
90 PRINT"TROVA QUESTO TASTO:  ";
100 FOR T=1 TO 500 : NEXT T
110 PRINT A$ : TI$="000000"
120 GET R$ : IF R$="" THEN 120
130 IF R$<>A$ THEN 120
140 POKE 214,14 : PRINT : POKE 211,6
150 PRINT "HAI IMPIEGATO";INT(TI/0.6)/10
0;
160 PRINT "SECONDI"
170 FOR T=1 TO 3000 : NEXT T
180 GOTO 10
READY.
```

Le righe da 10 a 60 cancellano lo schermo, visualizzano il titolo del programma, forniscono una pausa di circa due secondi e quindi il gioco ha inizio. La riga 70 sceglie una lettera a caso: questa operazione è effettuata attraverso CHR$, che converte un numero generato casualmente (compreso tra 65 e 90) in un singolo carattere assegnato ad A$. Le righe da 90 a 110 visualizzano un messaggio che vi richiede di localizzare la lettera casuale e poi azzerano il clock. Quando premete il tasto giusto, il valore del clock è letto e visualizzato dalla riga 150. Il calcolo:

```
INT (TI/0.6)/100
```

è usato per convertire il tempo in secondi e troncarlo a due cifre dopo il punto decimale. Se riuscite ad ottenere un punteggio inferiore a 0.5 secondi, siete probabilmente una dattilografa!!

MASCHERARE I BIT

A pagina 8 avete incontrato le due nuove parole chiave AND e OR e avete visto come possono essere utilizzate nella valutazione di condizioni logiche. C'è però dell'altro che riguarda AND e OR: entrambi sono in realtà esempi di "OPERATORI LOGICI", cioè parole chiave che combinano tra loro due numeri in un modo del tutto particolare. Per comprendere come il Commodore è programmato per produrre grafici ad alta risoluzione (i quali comportano l'accensione di punti e il tracciamento di linee), dovete conoscere come usare AND e OR per trasformare dei numeri in sequenze di punti sullo schermo.
Quando acquisite (PEEK) un valore da un registro, potete trattare il byte in esso contenuto come un singolo numero, disinteressandovi del fatto che si tratta, in realtà, di otto differenti bit; allo stesso modo potete immettere (POKE) un intero byte in un registro, così da modificare tutti i bit memorizzati nel registro stesso. Queste tecniche sono adatte per alcune applicazioni, ma presto vi accorgerete di avere la necessità di modificare un singolo bit nella memoria senza modificare nessuno degli altri bit di quel byte. Usando il Commodore questa tecnica è parte della programmazione sofisticata: per fare ciò dovrete utilizzare gli operatori AND e OR per la cosiddetta "mascheratura dei bit".

Come fare in modo che il Commodore modifichi un solo bit

Come ricorderete dal primo volume, V+21 (dove V=53248 è l'indirizzo di base del chip VIC) è il registro che controlla quali degli 8 sprite sono attivi e quali disattivi.

BIT DI CONTROLLO DEGLI SPRITE

Ogni sprite è attivato o disattivato da un singolo bit del registro V+21 del chip VIC.

Numero del bit	Numero dello Sprite controllato	Valore decimale del bit
0	0	1
1	1	2
2	2	4
3	3	8
4	4	16
5	5	32
6	6	64
7	7	128

Per essere in grado di controllare indipendentemente ogni singolo sprite, dovete necessariamente avere sotto controllo tutti i bit del registro V+21, potendoli definire a "1" o a "O" in un programma. Questo è quanto gli operatori AND e OR sono in grado di compiere: essi comparano due numeri bit per bit e producono un risultato basato su questa comparazione. L'operatore AND pone a livello logico "1" un bit del risultato solo nelle posizioni in cui esiste un bit a "1" in entrambi i numeri comparati. L'operatore OR, invece, pone a livello logico "1" un bit del risultato solo nella posizione in cui esiste un bit a "1" in almeno uno dei due numeri comparati.
Per fare un esempio, se volete porre a "1" il bit numero 4 di un byte, senza modificare il resto del byte stesso, potete programmare il computer per compiere la seguente operazione:

```
200 BYTE=BYTE OR 16
```

Il valore decimale del bit numero 4 di un byte è 16 (2↑4); se il byte ha valore 164, la precedente riga di programma effettuerà la seguente operazione:

COME FUNZIONA L'OPERATORE OR

L'operatore OR fornisce un bit a "1" quando uno o entrambi i bit che sta comparando sono a livello logico "1".

Numero del bit	7	6	5	4	3	2	1	0	
BYTE = 164 =	1	0	1	0	0	1	0	0	
16 =	0	0	0	1	0	0	0	0	OR
RISULTATO	1	0	1	1	0	1	0	0	

Il bit 4 inizialmente a livello logico "0" o "basso", è stato portato a livello "1" o "alto", mentre il resto del byte non è stato modificato.
Convertendo in binario il numero, gli effetti dell'OR diventano più chiari. Se, poi, lo convertite di nuovo in base dieci, esso diventerà 164 OR 16=180: un risultato che potete controllare visualizzandolo col computer. Per portare a livello "alto" o "accendere" uno o più bit in un byte, dovete applicare l'operatore OR a quel byte e a un numero dipendente dagli specifici bit che volete "accendere". Se invece volete portare a livello "basso" o "spegnere" dei particolari bit in un byte, dovete applicare l'operatore AND al byte stesso e a un numero ottenuto sottraendo a 225 i numeri relativi ai bit da "spegnere". Se volete "spegnere" il bit numero 5 nel valore 180 di prima dovete applicare l'operatore AND a 180 e al valore 225−32=223:

COME FUNZIONA L'OPERATORE AND

L'operatore AND fornisce un bit a "1" quando entrambi i bit che sta comparando sono a livello logico "1".

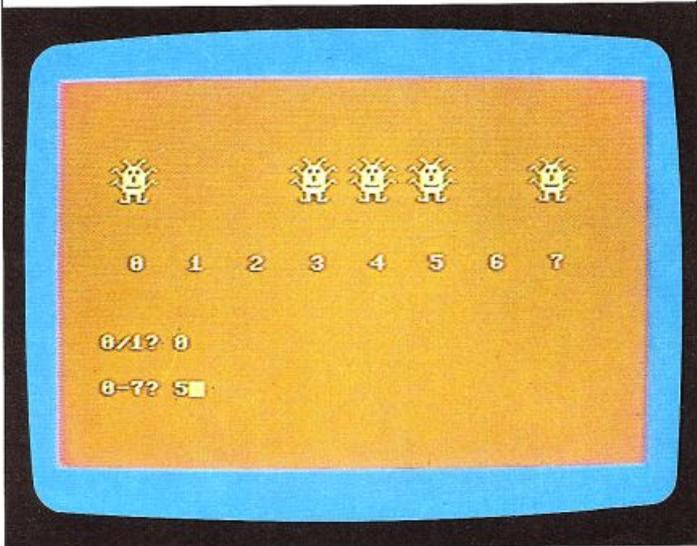
Numero del bit	7	6	5	4	3	2	1	0	
BYTE = 180 =	1	0	1	1	0	1	0	0	
255-32 = 223 =	1	1	0	1	1	1	1	1	AND
RISULTATO	1	0	0	1	0	1	0	0	

Attivare e disattivare gli sprite

Questo per quanto riguarda la teoria. Per vedere ora come Or e AND possono essere usati in un programma, consideriamo il programma seguente: esso utilizza il registro V+21 del chip VIC, insieme con un certo numero di INPUT dalla tastiera, per attivare o disattivare istantaneamente 8 sprites, secondo le vostre scelte:

COME USARE **AND** E **OR** CON GLI SPRITES

```
LIST -180

10 PRINT CHR$(147); : POKE 53280,6
20 POKE 53281,2 : FOR C=0 TO 62
30 READ BYTE
40 POKE 832+C,BYTE
50 NEXT C : V=53248 : POKE V+16,0
60 FOR C=0 TO 7
70 POKE 2040+C,13
80 POKE V+39+C,7
90 POKE V+2*C+1,90
100 HX=INT((32*C+54)/256)
110 POKE V+2*C,(32*C+54)-(HX*256)
120 POKE V+16,PEEK(V+16) OR (2↑C*HX)
130 POKE V+21,PEEK(V+21) OR 2↑C
140 NEXT C : PRINT CHR$(158);
150 POKE 214,10 : PRINT : POKE 211,1
160 FOR C=0 TO 7
170 PRINT TAB(C*4+4);C;
180 NEXT C : PRINT

READY.
```

```
LIST 190-

190 POKE 214,15 : PRINT : POKE 211,3
200 PRINT "0/1   ";
210 POKE 211,6 : INPUT A
220 POKE 214,18 : PRINT : POKE 211,3
230 PRINT "0-7";: POKE 211,6 : INPUT S
240 POKE 214,18 : PRINT : POKE 211,3
250 PRINT "
"
260 IF A=1 THEN 290
270 POKE V+21,PEEK(V+21) AND (255-2↑S)
280 GOTO 190
290 POKE V+21,PEEK(V+21) OR 2↑S
300 GOTO 190
800 DATA 3,0,192,0,129,0,0,66,0
810 DATA 0,126,0,96,255,6,145,153,137
820 DATA 11,153,208,7,255,224,3,231,192
830 DATA 3,231,192,99,231,198,147,255,20
1
840 DATA 11,126,208,7,0,224,3,255,192
850 DATA 3,255,192,0,255,0,0,195,0
860 DATA 0,195,0,4,195,32,7,195,224
READY.
```

Osservare la logica del computer al lavoro
Il programma seguente vi permetterà di visualizzare sullo schermo i risultati prodotti man mano dalle operazioni di "mascheratura" dei bit. Quando lo eseguite, esso vi richiede tre INPUT: il primo è la scelta tra l'operatore OR e l'operatore AND, mentre gli altri due sono numeri a cui verrà applicato l'operatore prescelto. Il programma converte poi in binario i vostri numeri decimali, esegue le operazioni necessarie e converte nuovamente il risultato in decimale.

PROGRAMMA "OPERATORI **AND** E **OR**"

```
10 PRINT CHR$(147);CHR$(5)
20 POKE 214,5 : PRINT : POKE 211,2
30 INPUT "VUOI VEDERE AND O OR  (A/O)";A
$
40 INPUT"        PRIMO VALORE (0-255)";V1
50 INPUT"     SECONDO VALORE (0-255)";V2
60 PRINT : PRINT TAB(4);V1;TAB(9);
70 V=V1 : GOSUB 500 : PRINT
80 PRINT TAB(4);V2;TAB(9)
90 V=V2 : GOSUB 500 : PRINT
100 IF A$="A" THEN 130
110 PRINT TAB(3);"=";V1 OR V2;TAB(9);
120 V=V1 OR V2 : GOSUB 500 : GOTO 150
130 PRINT TAB(3);"=";V1 AND V2;TAB(9);
140 V=V1 AND V2 : GOSUB 500
150 PRINT : PRINT : PRINT TAB(2);
160 INPUT "PREMI RETURN PER CONTINUARE";
A$
170 GOTO 10
500 FOR C=7 TO 0 STEP -1
510 PRINT SGN(V AND 2↑C);
520 NEXT C : PRINT : RETURN
READY.
```

Questo programma, alla riga 510, utilizza una funzione predefinita che non avete mai incontrato fino ad ora: si tratta della funzione SGN e si trova alla linea che effettua la conversione da decimale a binario. Il numero decimale è contenuto nella variabile V e il numero del bit da stampare risiede nella variabile C.
L'espressione:

V AND (2↑C)

fornisce il Cesimo bit del valore V. Se questo bit è "acceso" il risultato di questa espressione vale 2↑C, altrimenti vale O. Tuttavia, 2↑C e O, devono essere convertiti in 1 e 0 per poter essere stampati, ed è qui che entra in gioco la funzione SGN: essa fornisce il valore 1 se il suo argomento è positivo, -1 se è negativo e 0 se è 0. In questo programma essa fornirà solo due valori: 0 e 1:

IL RISULTATO DEL CALCOLATORE **AND/OR**

ALTA RISOLUZIONE

In tutti i disegni che avete visto fino ad ora sullo schermo, apparivano solo simboli predefiniti in bassa risoluzione, oppure sprite. Ma oltre a questi due modi grafici, il Commodore possiede anche una potente grafica ad alta risoluzione. Ora, dopo aver visto come funziona la "mascheratura" dei bit, siete in grado di predisporre uno schermo grafico ad alta risoluzione; nelle pagine seguenti vedremo come usarlo.

Mappa dello schermo
Lo schermo grafico ad alta risoluzione del Commodore è basato su una griglia rettangolare di 64000 pixel (punti dello schermo), controllabili individualmente e organizzati in 200 righe di 320 pixel ognuna (uno schermo grafico ad alta risoluzione è mostrato a pag. 58). Ogni pixel dello schermo è controllato da un bit della memoria: se il bit è a "1" allora il pixel è acceso, se invece è a "0", il pixel è spento.
I 64000 pixel richiedono dunque 64000 bit, ovvero 8000 byte di memoria per il loro controllo che viene effettuato dal chip VIC, il medesimo che gestisce gli sprite.
Per produrre disegni in alta risoluzione, occorre avere a disposizione un'area di memoria di 8000 byte; sfortunatamente l'area di memoria più adatta a questo scopo è già utilizzata per memorizzare i vostri programmi BASIC, dunque, essi non possono più rimanere memorizzati in questa parte della memoria e voi potete facilmente fare in modo che la ROM BASIC li memorizzi da qualche altra parte.

Come modificare l'area di memorizzazione del BASIC
Modificare l'area di memorizzazione dei programmi è molto semplice, come mostrano le schermate seguenti. (Se avete dei problemi seri con l'alta risoluzione, premete RUN/STOP e RESTORE contemporaneamente: questa operazione vi farà tornare in bassa risoluzione e potrete cercare eventuali errori nel vostro programma).

MODIFICA AREA MEMORIZZAZIONE PROGRAMMI

```
POKE 642,64
READY.
POKE 44,64
READY.
POKE 16384,0
READY.
NEW
READY.
```

E' molto importante ricordare che, prima di iniziare ad usare un programma con disegni in alta risoluzione, dovete sempre immettere questa serie di comandi. Se non lo fate, il vostro programma si interromperà malamente in quanto il computer tenterà di memorizzare il programma stesso e il disegno in alta risoluzione nella stessa zona di memoria. Tutti i programmi riportati nel resto di questo libro sono scritti supponendo che voi abbiate già immesso questi comandi.
Si ricordi che questa sequenza deve essere immessa mediante comandi diretti, non come parte di un programma, dal momento che tali comandi alterano il contenuto di alcuni puntatori di memoria utilizzati dal BASIC; se voi li inserite in un programma, il BASIC non riuscirà più a localizzare il resto del programma.
Proseguiamo. Tutti i programmi di aiuto per i disegni ad alta risoluzione che incontrerete in questo libro utilizzeranno una o più serie di subroutine (le incontrerete man mano nella lettura): i numeri di riga di ognuna di queste subroutine non si sovrappongono a quelli delle altre cosicché esse possono essere utilizzate come un unico programma. Se le memorizzate tutte assieme su un nastro o un disco, sarete sempre in grado di richiamare la routine che vi necessita.
Potete aggiungere ogni successiva routine all'insieme che avete salvato: immettete la nuova routine, caricate in memoria quello che avete salvato fino ad ora e quindi premete RETURN con il cursore posizionato ad ogni linea della nuova subroutine. Questa operazione unirà i due programmi. Non visualizzate (LIST) le routine precedenti altrimenti quella nuova sparirà dallo schermo senza alcuna possibilità di essere salvata.

Come azzerare la memoria
Le prime due subroutine che predispongono la pagina video in alta risoluzione, "dicono" al chip VIC dove trovare la memoria per la grafica ad alta risoluzione e come cancellare lo schermo azzerando parte della memoria.

AZZERAMENTO DELLA MEMORIA (SUBROUTINE 1-2)

```
LIST
10 GOSUB 100
20 COL=18 : GOSUB 200
30 END
100 POKE 53272,PEEK(53272) OR 8
110 POKE 53265,PEEK(53265) OR 32
120 RETURN
200 FOR MEM=8192 TO 16191
210 POKE MEM,0 : NEXT MEM
220 FOR MEM=1024 TO 2023
230 POKE MEM,COL : NEXT MEM
240 RETURN
READY.
```

Le righe 100 e 110 "dicono" al chip VIC di utilizzare il modo grafico in alta risoluzione e che gli 8000 byte di memoria necessari partono dalla locazione 8192. L'altra subroutine parte dalla riga 200. Le righe 200 e 210 cancellano lo schermo ponendo (POKE) a "0" tutti i bit degli 8000 bytes. Le righe 220 e 230 caricano in tutti i byte di un'altra area di memoria (dalla locazione 1024 alla locazione 2023) il valore di una variabile chiamata COL; questa zona della memoria contiene normalmente i codici dei caratteri alfanumerici e di quelli grafici predefiniti che appaiono sullo schermo, ma nella grafica ad alta risoluzione ha una nuova funzione: essa determina il colore dei vari pixel dello schermo. Dal momento che abbiamo 1000 bytes di memoria colore e 8000 bytes di memoria video, è chiaro che ogni byte di memoria colore controlla il colore di pixel acceso e pixel spento di 8 byte della memoria video.
In ogni byte di memoria per il colore, i bit da 4 a 7 controllano il colore di pixel acceso di un rettangolo di 8X8 pixel, mentre i bit da 0 a 3 controllano il suo colore di fondo (pixel spento); caricando, per mezzo dell'istruzione POKE e della variabile COL, valori opportuni nella memoria per il colore, potete creare qualsiasi combinazione di colori per i pixel accesi e spenti.

CODICE DEI COLORI IN ALTA RISOLUZIONE

Per selezionare una qualsiasi di queste combinazioni, i due codici di colore devono essere sommati assieme e quindi il risultato deve essere caricato nella memoria colore.

Colore	Codice pixel acceso	Codice pixel spento
Nero	0	0
Bianco	16	1
Rosso	32	2
Ciano	48	3
Porpora	64	4
Verde	80	5
Blu	96	6
Giallo	112	7
Arancione	128	8
Marrone	144	9
Rosa	160	10
Grigio scuro	176	11
Grigio	192	12
Verde chiaro	208	13
Azzurro	224	14
Grigio chiaro	240	15

Se volete disegnare in bianco su uno sfondo rosso, la combinazione dei codici colore dovrà essere 16+2, cioè 18; è quindi necessario caricare questo valore in tutte le 1000 locazioni della memoria colore, e questo è il compito delle righe 220 e 230 della precedente routine. L'azzeramento della memoria video e la predisposizione della memoria colore sono operazioni che richiedono tempo poichè coinvolgono l'esecuzione di 9000 POKE; l'unico modo per accelerarle è quello di usare il codice macchina, un'alternativa alla routine BASIC è mostrata alla fine della pagina. La prossima schermata mostra l'evoluzione di queste operazioni in BASIC: se usate il codice macchina, l'azzeramento avverrà così rapidamente da non poter vedere questa immagine:

L'ALTA RISOLUZIONE DURANTE L'INIZIALIZZAZIONE

Ora che avete ottenuto un programma che cambia l'area di memorizzazione del BASIC, predispone il chip VIC per l'alta risoluzione, cancella lo schermo e ne definisce i colori, siete veramente pronti per aggiungervi altre subroutine che faranno sì che il computer sia in grado di produrre dei disegni. Per sicurezza, vi conviene salvare (SAVE) le routine immesse fino ad ora; se non volete farlo, non battete NEW prima di continuare!

Predisposizione dell'alta risoluzione mediante codice macchina

La schermata seguente mostra una alternativa in codice macchina alla subroutine BASIC numero 2.
La sua numerazione è completamente compatibile con tutte le altre routine per l'alta risoluzione presentate in questo libro, cosicchè potete immetterla come una unità indipendente, senza dover capire come essa funziona.
La routine è in realtà costituita da un programma BASIC, il quale richiama una operazione in codice macchina usando il comando SYS.
Le istruzioni DATA e POKE sono utilizzate per cancellare dei valori in un certo insieme di locazioni di memoria le quali predispongono lo schermo ad alta risoluzione.

SUBROUTINE 2 (CODICE MACCHINA)

```
LIST

200 DATA 0,165,252,197,254,208,7,165
210 DATA 251,197,253,208,1,96,160,0
220 DATA 173,80,195,145,251,230,251
230 DATA 208,232,230,252,76,81,195
240 RESTORE : FOR C=50000 TO 50029
250 READ BYTE : POKE C,BYTE : NEXT C
260 POKE 251,0 : POKE 252,4 : POKE 253,2
32
270 POKE 254,7 : POKE 50000,COL : SYS 50
001
280 POKE 251,0 : POKE 252,32 : POKE 253,
64
290 POKE 254,63 : POKE 50000,0 : SYS 500
01 : RETURN

READY.
█
```

DISEGNARE CON I PUNTI

Con tutti i microcomputer i disegni vengono effettuati accendendo una specifica serie di pixel. Per accendere un singolo pixel col Commodore, dovete innanzitutto scoprire quale delle 8000 locazioni della memoria video controlla il pixel da illuminare e, infine, quale bit all'interno di tale locazione deve essere portato a livello alto. Dal momento che i 64000 pixel sono sistemati su 200 righe di 320 colonne, ogni punto dello schermo può essere specificato da un numero di riga e uno di colonna, come un qualsiasi carattere sulla pagina di testo. I pixel sono numerati da 0 a 199 verso il basso e da 0 a 319 verso destra, cosicché il pixel con numero più basso (contenuto nella locazione di memoria 8192) si viene a trovare nell'angolo in alto a sinistra dello schermo.

MEMORIA VIDEO IN ALTA RISOLUZIONE

La pagina video in alta risoluzione è composta da 64000 punti controllabili separatamente e numerati da 0 a 319 in orizzontale e da 0 a 199 in verticale. La tabella mostra solo i byte che controllano la parte superiore sinistra dello schermo (la tabella completa è a pag. 58)

Coordinate orizzontali / Coordinate verticali

	0-7	8-15	16-23	24-31	→	312-319
0	8192	8200	8208	8216	→	8504
1	8193	8201	8209	8217	→	8505
2	8194	8202	8210	8218	→	8506
3	8195	8203	8211	8219	→	8507
4	8196	8204	8212	8220	→	8508
5	8197	8205	8213	8221	→	8509
6	8198	8206	8214	8222	→	8510
7	8199	8207	8215	8223	→	8511
8	8512	8520	8528	8536	→	8824
9	8513	8521	8529	8537	→	8825
10	8514	8522	8530	8538	→	8826
↓	↓	↓	↓	↓		
199	15879	15887	15895	15903		16191

Per memorizzare tutte le informazioni relative all'intera pagina video per poter modificare un particolare byte, sarebbe richiesta una enorme quantità di memoria; fortunatamente, potete evitare l'inconveniente usando due equazioni: la prima dice in quale byte si trova il pixel cercato date le sue coordinate; la seconda fornisce il valore di "mascheratura" per quel bit; potete usare quei valori per predisporre un particolare bit con la tecnica di mascheratura delle pagg. 12-13.

Queste due equazioni sono pronte per l'uso nel prossimo gruppo di subroutine; dovrete immetterle sul video, caricare (LOAD) il primo gruppo di routine (riportato a pag. 14) e quindi usare il tasto RETURN, con il cursore posizionato a ogni nuova linea, per unire i due gruppi di routine.

Non è necessario che capiate esattamente come funziona la subroutine: sono solo calcoli per identificare un certo bit ed accenderlo, o spegnerlo, come richiesto. Il "disegnatore di punti" permette di modificare dei pixel, utilizzando una unica istruzione di POKE all'interno di un ciclo, anzichè più istruzioni di POKE separate:

DISEGNATORE DI PUNTI (SUBROUTINE DA 3 A 5)

```
LIST

300 BYTE=8192+INT(LY/8)*320+INT(LX/8)*8+
(LY AND 7)
310 MASK=2↑(7-(LX AND 7))
320 RETURN
400 GOSUP 300
410 POKE BYTE,PEEK(BYTE) OR MASK
420 CMEM=1024+INT(LY/8)*40+INT(LX/8)
430 POKE CMEM,COL
440 RETURN
500 GOSUB 300
520 POKE BYTE,PEEK(BYTE) AND (255-MASK)
530 RETURN

READY.
█
```

Il "disegnatore di punti" contiene tre differenti subroutine. La prima, che inizia alla riga 300, calcola il byte e la maschera relativi al pixel che si trova alle coordinate LX o LY, ed è richiamata da altre due: la prima di queste alla riga 400, usa questi calcoli e il valore della variabile COL per illuminare il pixel alle coordinate LX ed LY nel colore prescelto mentre l'altra, che inizia alla riga 500, spegne il pixel alle coordinate LX ed LY, sempre calcolate dalle subroutine alla linea 300.

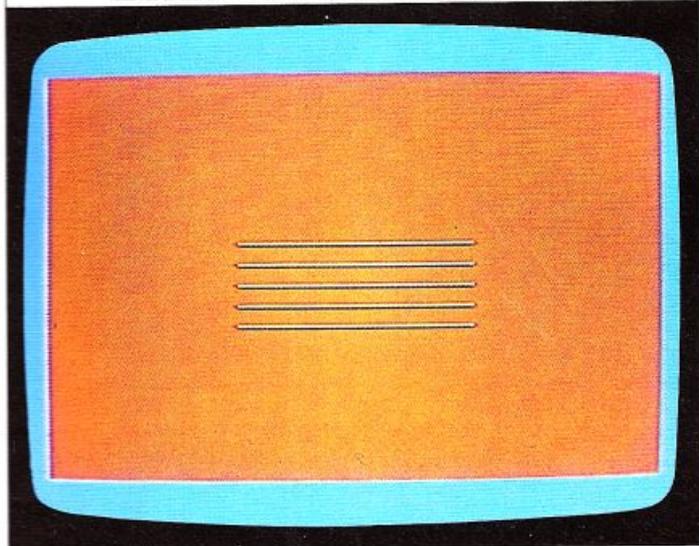
A questo punto vi conviene "salvare" nuovamente queste routines dopo averle aggiunte alle due precedenti, dal momento che esse saranno usate frequentemente nelle pagine seguenti.

Come tracciare dei disegni

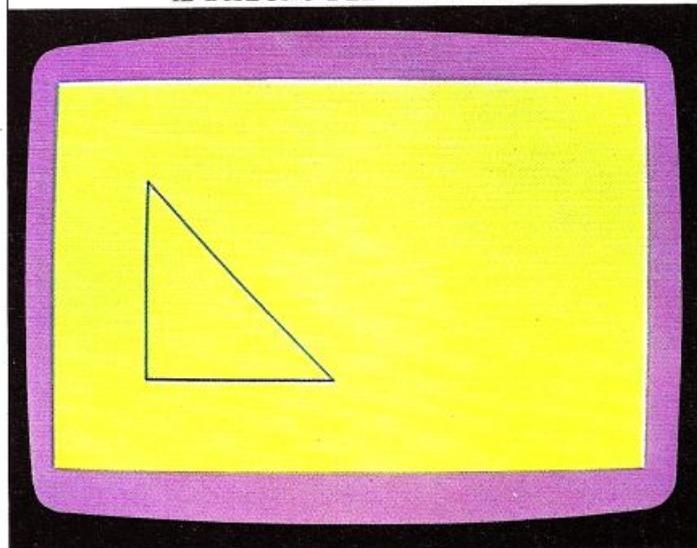
Ora, dando per scontato che il vostro Commodore ha in memoria un totale di cinque differenti subroutine, immettete ed eseguite il seguente programma, il quale le farà entrare in azione:

COME DISEGNARE LINEE PARALLELE

La riga 10 predispone lo schermo in alta risoluzione. La riga 20 inizializza COL a 18 per ottenere linee bianche su uno sfondo rosso ed, inoltre, cancella lo schermo. La riga 50 richiama la subroutine che accende un pixel, ed è contenuta in due cicli innestati di FOR...NEXT (righe dalla 30 alla 70): questi due cicli generano i valori di LX ed LY in modo tale che il programma disegni cinque linee orizzontali costituite da una sequenza di singoli pixel:

IL DISEGNO DELLE LINEE PARALLELE

L'istruzione END alla riga 80 è necessaria a causa delle subroutine che seguono il programma vero e proprio: senza END, esse verranno interpretate come una parte del programma ed eseguite alla fine del ciclo, distruggendo i risultati ottenuti. Il prossimo programma mostrerà come unire tra loro più linee usando sempre le cinque subroutine. Le linee verranno disegnate anche verticalmente e diagonalmente. Dal momento che tale programma è più lungo di nove righe, esso non potrà essere infilato nei numeri di riga precedenti le subroutine (da 0 a 90), come nel caso precedente; per questo, al programma sono stati assegnati numeri di riga maggiori di quelli delle subroutine, a cominciare da 1000:

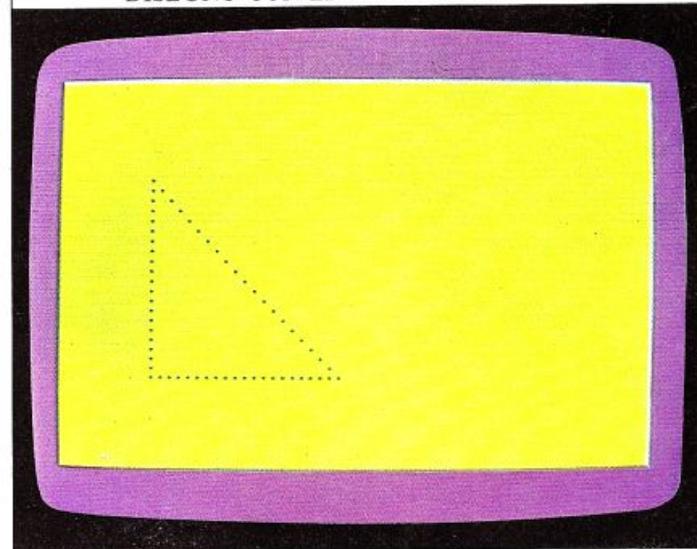
PROGRAMMA "TRIANGOLO"

IL DISEGNO DEL TRIANGOLO

Come cambiare il passo in un ciclo

Avrete certamente notato che, nei programmi precedenti, le linee sono costituite da una sequenza di pixel contigui: questo effetto è stato ottenuto per mezzo del ciclo FOR...-NEXT. Voi, però, potete modificare tali programmi in modo tale che, invece di linee continue, essi producano linee tratteggiate: è sufficiente impostare il passo (STEP) dei cicli in modo che essi tralascino qualche pixel. Potete ottenere questo effetto nel programma "triangolo" effettuando le seguenti modifiche:

```
1030 FOR LY=50 TO 150 STEP 5
1060 FOR LX=50 TO 150 STEP 5
1080 FOR C=150 TO 50 STEP -5
```

Il disegno è ora costituito da singoli pixel accesi, ognuno separato da cinque pixel spenti (potete usare questa tecnica per disegnare linee tratteggiate all'interno delle figure). In ogni caso il contorno è lo stesso:

DISEGNO CON LINEE TRATTEGGIATE

Siccome il programma ora deve accendere soltanto un quinto dei pixel, esso è considerevolmente più veloce.

COME DISEGNARE LINEE

Ora voi sapete come accendere i pixel sullo schermo e come disegnare linee e semplici figure usando il ciclo FOR...NEXT. In ogni caso, usando le tecniche descritte nelle precedenti pagine, siete in grado di disegnare solo linee verticali, orizzontali oppure inclinate di 45° mentre, per costruire dei disegni, avete bisogno di un modo per ottenere linee con una angolazione qualsiasi, e questo è quanto state per scoprire: una subroutine che può disegnare una linea tra due punti qualsiasi dello schermo.

La subroutine che disegna

La subroutine di base per il disegno di linee compare nella schermata seguente. Dovrete aggiungerla alle varie subroutines che conservate su nastro o su disco in modo da averne, ora, sei in tutto:

DISEGNO DI LINEE (SUBROUTINE 6)

```
LIST

600 GT=ABS(NX-LX)
610 IF ABS(NY-LY)>GT THEN GT=ABS(NY-LY)
620 XINC=(NX-LX)/GT : YINC=(NY-LY)/GT
630 XX=LX+0.5 : YY=LY+0.5
640 FOR CC=1 TO GT
650 LX=INT(XX) : LY=INT(YY) : GOSUB 400
660 XX=XX+XINC : YY=YY+YINC
670 NEXT CC : LX=NX : LY=NY : RETURN
READY.
```

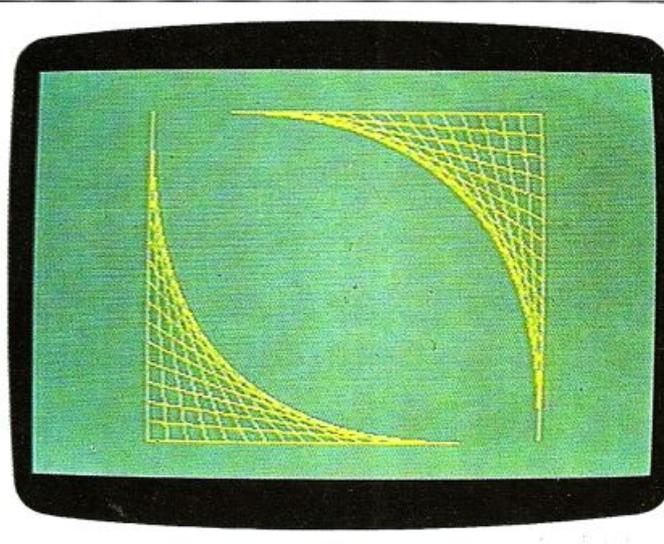
Non preoccupatevi se non riuscite a capire completamente le precedenti righe di programma: la subroutine è semplicemente un insieme di equazioni che calcolano la posizione dei pixel da accendere. Essa traccia una linea sullo schermo dal pixel di coordinate LX, LY a quello di coordinate NX,NY e quindi aggiorna le coordinate di partenza (LX, LY) rendendole uguali a quelle di arrivo (NX, NY), in modo che un'altra linea possa essere tracciata a partire dalla posizione finale della precedente.

La parola chiave ABS, che compare alle righe 600 e 610, fornisce il valore assoluto di ogni costante o variabile numerica che la segue; in questa subroutine essa ha il compito di rendere positivo qualsiasi valore di NX–LX e di NY–LY, per permettere al programma di utilizzare solo valori positivi per il tracciamento delle linee.

Ricami a "spago e spillo"

Ora che disponete di sei subroutine, provate ad aggiungervi le righe di programma seguenti e, quindi, lanciate il programma completo per vedere che figura apparirà:

PROGRAMMA "SPAGO E SPILLO"

```
LIST

10 GOTO 1000
1000 POKE 53280,11 : GOSUB 100
1010 COL=124 : GOSUB 200
1020 FOR C=0 TO 160 STEP 10
1030 LY=20 : NX=260
1040 LX=C+100 : NY=C+20
1050 GOSUB 600
1060 NEXT C
1070 FOR C=0 TO 160 STEP 10
1080 LX=60 : NY=180 : COL=124
1090 LY=C+20 : NX=C+60
1100 GOSUB 600

READY.
```

E' molto semplice produrre questo tipo di reticolo in quanto esiste una banale relazione matematica tra i punti in cui terminano le varie linee: se cambiate i valori usati, otterrete un disegno differente. Ma cosa succederebbe se voleste disegnare qualcosa che non è descrivibile da una semplice equazione.... uno space shuttle, per esempio? Per fare ciò avete bisogno di una subroutine che tracci linee che potete specificare individualmente, in modo da ottenere la figura che desiderate.

Una macchina per tracciare linee in alta risoluzione

Per memorizzare le informazioni relative a ogni singola linea, potete usare lo stesso metodo che impieghereste per memorizzare le note in un programma di generazione musicale: un insieme di istruzioni DATA. Nel programma seguente, i valori nelle istruzioni DATA descrivono un certo disegno: potete pensare che questo programma simuli una "penna" immaginaria e, come una penna reale, potete sollevarla dallo schermo, riappoggiarla e muoverla in una direzione qualsiasi:

DISEGNARE CON L'ISTRUZIONE DATA

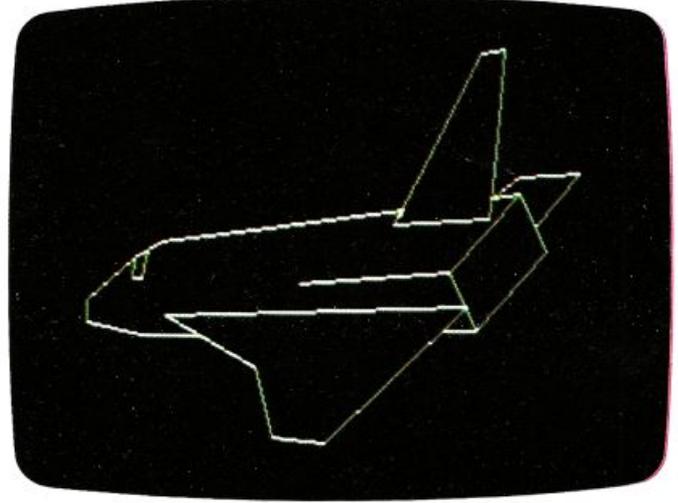
```
LIST

10 GOTO 1000
1000 POKE 53280,0 : GOSUB 100
1010 COL=208: GOSUB 200
1020 PEN=0 : LX=0 : LY=0
1030 READ NX,NY
1040 IF NX>=0 THEN 1090
1050 IF NY=1 THEN PEN=1 : GOTO 1030
1060 IF NY=0 THEN PEN=0 : GOTO 1030
1070 IF NY=2 THEN 1070
1080 COL=-NY : GOTO 1030
1090 IF PEN=0 THEN LX=NX : LY=NY : GOTO
1030
1100 GOSUB 600 : GOTO 1030

READY.
```

```
LIST 1200-

1200 DATA -1,0,49,123,-1,1,74,102,73,112
 78,112,81,98,74,102,88,94,212,78
1210 DATA 207,85,254,4,266,2,256,82,207,
85,-1,0,258,72,-1,1
1220 DATA 272,70,286,98,250,137,236,108,
272,70
1230 DATA 258,72,272,62,304,60,280,86,-1
 0,245,125,-1,1
1240 DATA 90,130,136,156,145,192,173,195
 245,125
1250 DATA -1,0,236,108,-1,1,158,114,-1,0
 106,140,-1,1,78,140,49,134,49,123
1260 DATA -1,0,250,137,-1,1,233,137
1360 DATA -1,2
READY.
```

Il programma vi fornisce anche la possibilità di cambiare il colore dell'inchiostro, e questa operazione è controllata ancora da istruzioni DATA, le quali permettono di modificare il valore della variabile COL.

Come funziona la macchina per tracciare le linee
In questo programma tutti i valori contenuti nelle righe di DATA sono raggruppabili a coppie: se entrambi i valori sono maggiori o uguali a zero, la penna si sposta in linea retta verso la posizione specificata dalla coppia di coordinate e, se essa è appoggiata sullo schermo, viene visualizzata una linea. Per tutte le altre operazioni possibili, il primo valore della coppia è −1 mentre il secondo specifica al computer cosa fare. Se il secondo numero è 1, la penna viene appoggiata sullo schermo; se è 0, essa viene sollevata; se tale numero è negativo, il suo valore assoluto viene assegnato alla variabile COL e specifica, quindi, il colore dell'inchiostro. Infine, se il secondo numero è 2, il programma termina entrando in un ciclo senza fine alla riga 70: il programma termina in questo strano modo per evitare che il disegno ottenuto venga rovinato dal messaggio di READY che apparirebbe altrimenti sullo schermo. Per interromperlo definitivamente dovete solo premere il tasto di RUN/STOP.

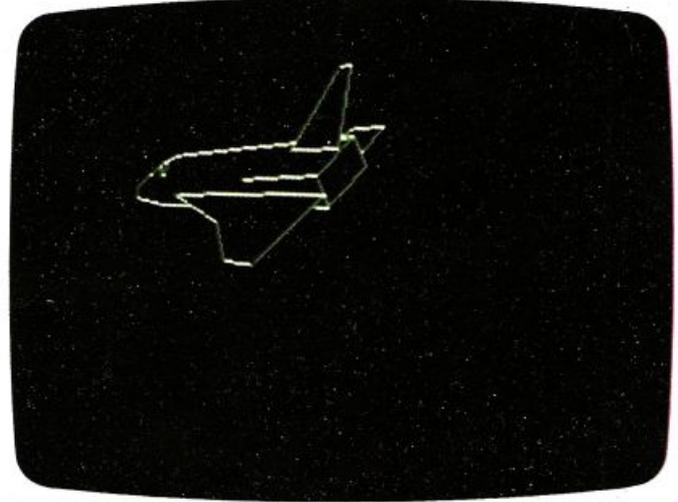
Con i valori contenuti nelle istruzioni DATA della schermata precedente, il programma produrrà il seguente disegno:

DISEGNO OTTENUTO DALLA MACCHINA TRACCIA LINEE

Ricordate che per LISTare il programma per modificarlo, nel caso abbiate commesso degli errori durante la sua immissione, dovete ritornare in bassa risoluzione: potete fare ciò premendo contemporaneamente i tasti RUN/STOP e RESTORE per riportare il vostro Commodore alla situazione precedente l'esecuzione del programma.

Come cambiare la dimensione del disegno
Potete modificare il programma "macchina per tracciare linee" in modo tale da ottenere lo stesso disegno ma in scala diversa.
Provate ad immettere le seguenti righe di programma:

```
600 NX=NX/2 : NY=NY/2 :GT=ABS(NX-LX)
1090 IF PEN=O THEN LX=NX/2 : LY=NY/2 :
GOTO 1030
```

Ciò ridurrà il disegno dello space shuttle alla metà della sua dimensione originale:

DISEGNO RIDOTTO DELLO SPACE SHUTTLE

CURVE E CERCHI

Ora che conoscete come ottenere disegni mediante l'uso di linee rette, potete ottenere che il vostro computer tracci anche cerchi e curve qualsiasi. Sul Commodore non esiste una istruzione CIRCLE che vi possa aiutare a questo proposito; dovrete invece utilizzare due nuove parole chiave: SIN e COS. Usando questi due nuovi comandi, potete ottenere dei disegni di grande effetto mediante programmi molto brevi.

Come il Commodore disegna un cerchio
In un arco di circonferenza, ogni punto sulla circonferenza è in relazione con un angolo al centro.

COORDINATE CIRCOLARI

È possibile esprimere le distanze X e Y usando il valore del seno (SIN) e del coseno (COS) dell'angolo al centro. Ogni angolo ha i propri valori di seno e coseno; le coordinate di ogni punto della circonferenza sono date da:

R*COS(A),R*SIN(A)

Una volta apprese queste relazioni, potete lanciare il vostro Commodore tra curve e cerchi: il programma seguente disegna un cerchio con un raggio di 80 pixel. Ricordate di spostare l'area di memoria BASIC (se non lo avete già fatto) e di caricare (LOAD) le sei routine per la grafica in alta risoluzione prima di eseguirlo.

PROGRAMMA "CERCHIO"

```
LIST

10 GOTO 1000
1000 COL=16 : POKE 53280,8 : GOSUB 100 :
 GOSUB 200
1010 FOR A=0 TO 2*π STEP π/120
1020 LX=80*COS(A)+160
1030 LY=80*SIN(A)+100
1040 GOSUB 400
1050 NEXT A
1060 GOTO 1060
READY.
```

Come mai il computer non usa i gradi
In questo primo programma l'angolo doveva variare tra 0° ed un angolo giro, cioè 360° ma, come avete probabilmente notato, il numero 360 non compare mai nel programma e, invece, il ciclo va da 0 a 2*π con uno strano passo di π/120.
La ragione di tutto ciò è che il Commodore, come la maggior parte dei computer, non usa affatto i gradi sessagesimali per misurare gli angoli, ma i radianti: un modo differente ma più logico per fare la stessa cosa. Un angolo giro di 360° è perfettamente equivalente a 2*π radianti.
Il simbolo π (pi greco) è una lettera dell'alfabeto greco che può essere riprodotta dal Commodore premendo SHIFT e il tasto vicino a RESTORE; si tratta di un'importantissima costante matematica che ha il valore di 3.14159265... (potete visualizzare questo numero mediante l'istruzione PRINT π).
Questo valore è il rapporto tra la lunghezza di una circonferenza ed il suo diametro.
Dovrete ricordarvi che ci sono 2*π radianti in un cerchio, cosicché π/2 radianti sono un quarto di cerchio, π/4 un ottavo e così via.

Una routine generalizzata per il cerchio
Potete ora aggiungere una nuova subroutine alle sei che avete già memorizzato, così il vostro Commodore potrà disegnare qualsiasi cerchio desideriate.
Questa subroutine è molto simile al programma precedente fatta eccezione per il fatto che, invece di disegnare solamente dei punti della circonferenza, essa li unisce con delle linee in modo da ottenere un contorno completo. Questa nuova subroutine comincia alla riga 700. Per poterla utilizzare dovete fornire al computer tre valori; essi costituiscono le coordinate del centro (XC,YC) e la lunghezza, in pixel, del raggio del cerchio (RAD):

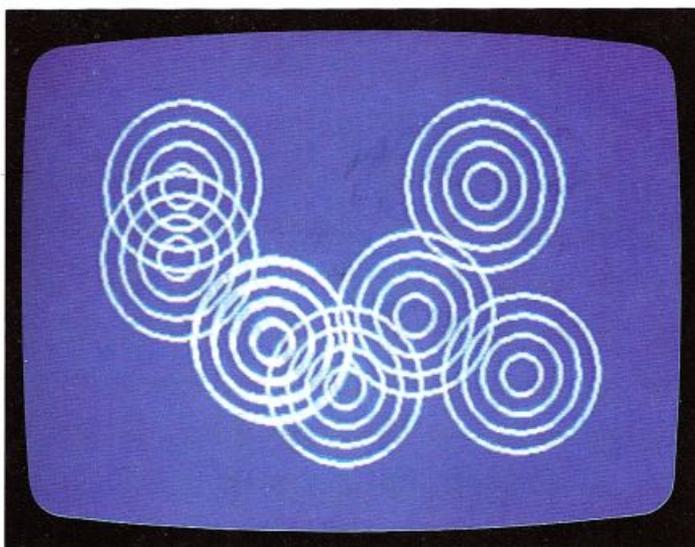
CIRCONFERENZA (SUBROUTINE 7)

Quando avrete aggiunto la precedente al gruppo delle sei subroutine che possedete già, memorizzatele tutte

assieme per poterle utilizzare poi. L'insieme di routine grafiche in alta risoluzione è, ora, quasi completo.

Costruire un ricamo con la subroutine per i cerchi
Se selezionate a caso le coordinate del centro di un cerchio, potete far costruire al computer dei ricami:

PROGRAMMA "CERCHI CONCENTRICI"

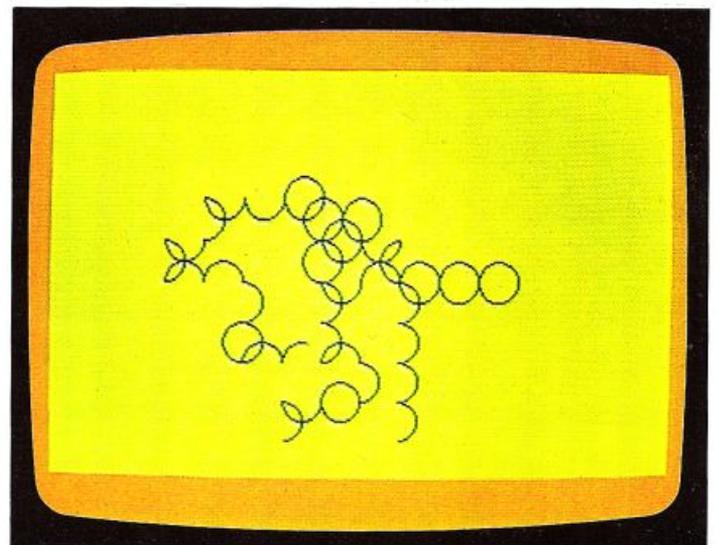
```
LIST

10 GOTO 1000
1000 COL=22 : POKE 53280,6 : GOSUB 100 :
 GOSUB 200
1010 FOR C=1 TO 8
1020 XC=50+INT(RND(0)*220)
1030 YC=50+INT(RND(0)*100)
1040 FOR RAD=10 TO 40 STEP 10
1050 GOSUB 700
1060 NEXT RAD : NEXT C
1070 GOTO 1070
READY.
```

Le righe 1020 e 1030 forniscono un paio di coordinate (X,Y) casuali in modo tale che nessuna di queste sia più vicina di 50 pixel ai bordi dello schermo: questa limitazione dipende dal fatto che il programma può disegnare cerchi con un raggio massimo di 50 pixel. Il ciclo contenuto nelle righe da 1040 a 1060 disegna un certo numero di cerchi concentrici con un raggio che cresce man mano; la riga 1060 fa ripartire l'intero processo, ma con un nuovo paio di coordinate casuali.
Potete provare a modificare il raggio massimo dei cerchi e il passo (STEP) tra le misure dei raggi, cambiando i valori nella riga 1040 in:

```
1040 FOR RAD=10T040 STEP 6
```

o, perfino, con STEP 3, il che produrrà ricami più piccoli e più ravvicinati.

Come programmare "curve vagabonde"
Quando avete immesso la routine per i cerchi, potreste aver pensato che questa sembri più complicata di quanto non lo sia realmente; e avreste avuto proprio ragione. Infatti non solo questa routine è in grado di disegnare dei cerchi, ma traccia anche archi e parti di cerchio: per utilizzare la subroutine per la tracciatura degli archi, dovete fornire al computer i valori di XC, YC e RAD, come per i cerchi, ma è necessario specificare anche i valori di due ulteriori variabili, A1 e A2, le quali controllano gli angoli iniziali e finali tra cui dovrà essere compreso l'arco. Questi angoli sono misurati in radianti in senso orario, a partire dal semiasse positivo delle X. Quando utilizzerete questa routine per disegnare degli archi, dovrete richiamarla con GOSUB 710 invece di GOSUB 700.
Il seguente è un programma che utilizza GOSUB 710 per disegnare dei semicerchi in posizione casuale, i quali possono essere diretti verso l'alto, il basso, a destra o a sinistra dello schermo; il programma seleziona un numero a caso tra 1 e 4 compresi, quindi utilizza questo numero per impostare la direzione in cui sarà tracciato un semicerchio:

PROGRAMMA "CURVE VAGABONDE"

```
LIST

10 GOTO 1000
1000 COL=103 : RAD=10
1010 POKE 53280,8 : GOSUB 100 : GOSUB 20
0
1020 LX=160 : LY=100
1030 Q=INT(RND(0)*4)+1
1040 IF Q=1 THEN XC=LX : YC=LY-10 : A1=π
/2
1050 IF Q=2 THEN XC=LX+10 : YC=LY : A1=π

1060 IF Q=3 THEN XC=LX : YC=LY+10 : A1=3
*π/2
1070 IF Q=4 THEN XC=LX-10 : YC=LY : A1=0

1080 IF XC>300 OR XC<20 OR YC>180 OR YC<
20 THEN 1020
1090 A2=A1+π
1100 GOSUB 710
1110 GOTO 1030
READY.
```

COLORARE LE FIGURE

Fino ad ora avete "assiemato" una "scatola di montaggio" per la grafica, molto completa, sotto forma di sei subroutine le quali possono disegnare un certo insieme di figure. Ora siete in grado di completare il vostro kit aggiungendovi l'ultima routine, la quale colora le figure che potete disegnare con le altre. Essa può funzionare con quasi tutte le figure chiuse, colorandole completamente con un dato colore; la parola "chiuse" significa che la figura deve essere contornata da una linea ininterrotta di pixel accesi: un cerchio costruito per punti, per esempio, non potrà essere colorato in questo modo, ma uno costruito mediante linee spezzate sì.

La subroutine che colora le figure
Per incominciare, ecco la subroutine che dovete aggiungere al precedente gruppo di sette:

INCHIOSTRATORE DI FIGURE (SUBROUTINE 8)

```
LIST -930

800 SP=0 : FU=0 : FD=0
810 FX=FX-1 : LX=FX : LY=FY : GOSUB 300
820 IF (PEEK(BYTE) AND MASK)=0 THEN 810
830 FX=FX+1 : LX=FX : GOSUB 400
840 LY=LY-1 : GOSUB 300
850 IF ((PEEK(BYTE) AND MASK)=0) AND (FU
=0) THEN FU=1 : GOSUB 940
860 IF ((PEEK(BYTE) AND MASK)<>0) AND (F
U=1) THEN FU=0
870 LY=LY+2 : GOSUB 300
880 IF ((PEEK(BYTE) AND MASK)=0) AND (FD
=0) THEN FD=1 : GOSUB 940
890 IF ((PEEK(BYTE) AND MASK)<>0) AND (F
D=1) THEN FD=0
900 LY=LY-1 : LX=LX+1 : GOSUB 300
910 IF (PEEK(BYTE) AND MASK)=0 THEN 830
920 IF SP=0 THEN RETURN
930 GOSUB 970 : GOTO 810

READY.
```

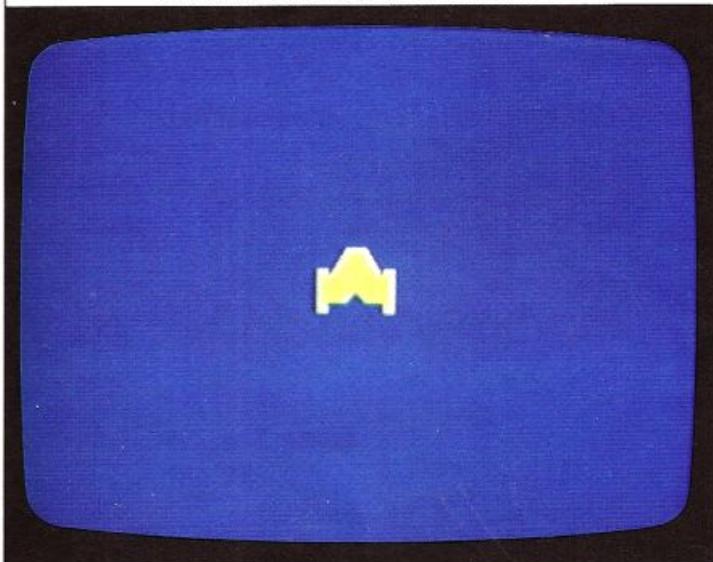
```
LIST 940-

940 IF SP>9 THEN PRINT "FIGURA TROPPO CO
MPLESSA DA COLORARE" : END
950 ST(SP,0)=LX : ST(SP,1)=LY : SP=SP+1
960 RETURN
970 FU=0 : FD=0 : SP=SP-1
980 FX=ST(SP,0) : FY=ST(SP,1)
990 RETURN
READY.
```

Per utilizzare questa routine, dovete predisporre due variabili, FX e FY, le quali contengono due coordinate che si devono trovare all'interno dell'area che volete colorare, non importa in quale parte della stessa. Una volta fatto ciò, richiamate la subroutine con GOSUB 800, ed essa vi colorerà l'area che avete scelto.
È bene, ora, fare una precisazione: durante questi esperimenti avrete notato che il Commodore disegna punti e linee piuttosto lentamente quando usate il BASIC. La subroutine che colora non è più veloce di quella che traccia linee e, siccome ci sono moltissimi pixel da accendere, il colorare le figure richiede molto tempo; comunque, come potrete vedere, il risultato ripaga ampiamente l'attesa.

Come funziona l'inchiostratore di figure
Benché la routine sembri complicata, quello che in realtà essa esegue è semplice. Le righe 810 e 820 spostano la coordinata FX alla sinistra del suo valore iniziale finché non viene rilevato il contorno della figura. Le righe da 830 a 890 ripetono lo stesso percorso da sinistra a destra accendendo dei pixel, così da tracciare una linea orizzontale. Mentre vengono accesi dei punti, le righe da 850 a 890 controllano anche i pixel al di sotto e al di sopra per vedere se questi dovranno essere accesi più tardi. Se è necessario accendere i pixel sopra e sotto la linea tracciata, le coordinate dell'estrema sinistra delle linee a cui appartengono tali pixel vengono memorizzate alle righe 940 e 960; tali coordinate sono caricate in un "array", un metodo per memorizzare informazioni in modo che si possa accedere separatamente ad ogni termine (gli "array" sono trattati alle pagg. 52 e 53).
Quando viene raggiunto l'estremo opposto della linea corrente (cioè quando la routine ha incontrato il margine destro della figura), vengono recuperate un paio di coordinate dell'array e il processo ricomincia, partendo da queste nuove coordinate.
Per osservare l'inchiostratore all'opera, immettete il programma seguente alla fine dell'insieme completo delle otto subroutine grafiche e quindi eseguitelo. Ricordatevi che, se non lo avete già fatto, dovete modificare l'area di memoria BASIC come al solito:

COLORARE UN VEICOLO LUNARE

```
LIST
10 GOTO 1000
1000 COL=118 : GOSUB 100 : GOSUB 200 : P
OKE 53280,6
1010 LX=155 : LY=80
1020 FOR C=1 TO 17
1030 READ NX,NY : GOSUB 600
1040 NEXT C
1050 FX=160 : FY=90 : GOSUB 800
1060 GOTO 1060
1070 DATA 165,80,175,95,175,90
1080 DATA 180,90,180,110,175,110
1090 DATA 175,105,165,105,160,100
1100 DATA 155,105,145,105,145,110
1110 DATA 140,110,140,90,145,90
1120 DATA 145,95,155,80
READY.
```

FIGURA DEL VEICOLO LUNARE

In questo programma, la riga 1000 decide per una figura in giallo su di uno sfondo blu, predispone il computer per l'alta risoluzione e cancella lo schermo. Le righe da 1010 a 1040 interpretano il contenuto delle sei istruzioni DATA come coppie di coordinate e disegnano il contorno del veicolo lunare. La riga 1050, inizializza FX e FY con le coordinate di un punto all'interno del contorno, mentre la riga 1060 impedisce che i messaggi di testo rovinino il disegno terminato.

Come colorare il disegno dello space shuttle

Il programma seguente colora lo space shuttle disegnato a pag. 19 e vi aggiunge il disegno di una luna (per mezzo delle righe da 1120 a 1150). Le coordinate contenute nelle righe da 1110 a 1116 specificano quale parte dello shuttle deve essere colorata. In questo programma sono specificate quattro paia di coordinate: un paio per i finestrini della cabina, due per le ali e uno per la coda. L'inchiostratura della luna è specificata alla riga 1160 ed essa è colorata con un inchiostro di colore differente, predisposto alla riga 1120. Anche questo programma termina con un ciclo senza fine:

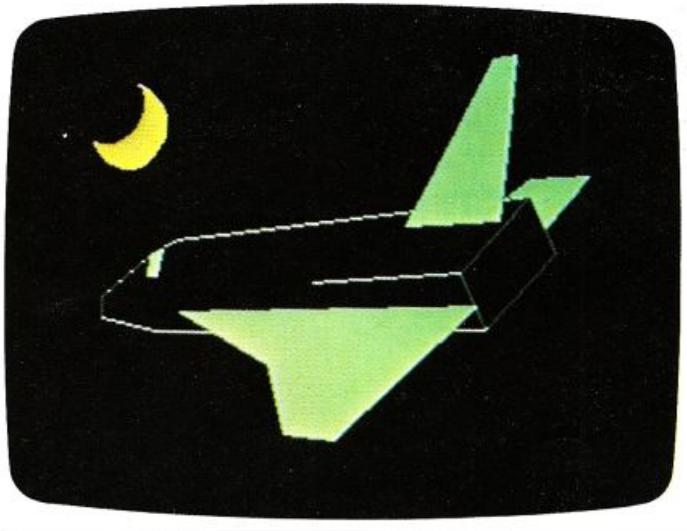
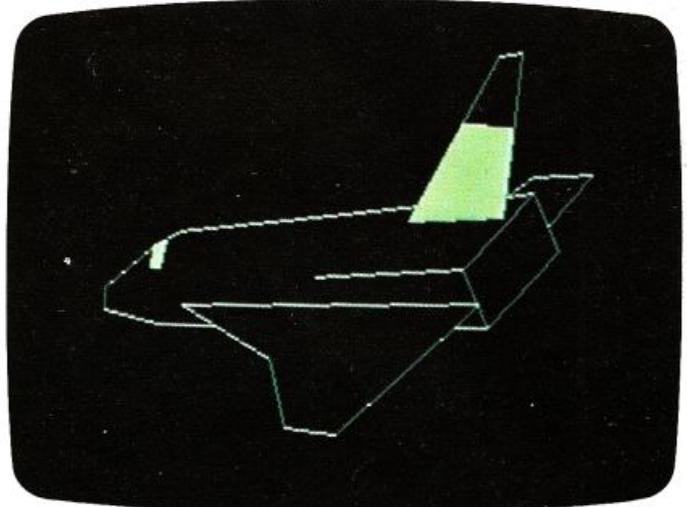
INCHIOSTRATORE DELLO SPACE SHUTTLE

```
LIST -1140

10 GOTO 1000
1000 GOSUB 100
1010 COL=208 : GOSUB 200
1020 PEN=0 : LX=0 : LY=0
1030 READ NX,NY
1040 IF NX>=0 THEN 1090
1050 IF NY=1 THEN PEN=1 : GOTO 1030
1060 IF NY=0 THEN PEN=0 : GOTO 1030
1070 IF NY=2 THEN 1110
1080 COL=-NY : GOTO 1030
1090 IF PEN=0 THEN LX=NX : LY=NY : GOTO
1030
1100 GOSUB 600 : GOTO 1030
1110 FX=75 : FY=110 : GOSUB 800
1112 FX=220 : FY=80 : GOSUB 800
1114 FX=173 : FY=194 : GOSUB 800
1116 FX=280 : FY=70 : GOSUB 800
1120 COL=112 : XC=60 : YC=40 : RAD=20
1130 A1=4.9 : A2=3 : GOSUB 710
1140 XC=48 : YC=29 : RAD=20

READY.
```

INCHIOSTRATORE DELLO SPACE SHUTTLE

```
LIST 1150-

1150 A1=5.8 : A2=2 : GOSUB 710
1160 FX=70 : FY=50 : GOSUB 800
1170 GOTO 1170
1200 DATA -1,0,49,123,-1,1,74,102,73,112
,78,112,81,98,74,102,88,94,212,78
1210 DATA 207,85,254,4,266,2,256,82,207,
85,-1,0,258,72,-1,1
1220 DATA 272,70,286,98,250,137,236,108,
272,70
1230 DATA 258,72,272,62,304,60,280,86,-1
,0,245,125,-1,1
1240 DATA 90,130,136,156,145,192,173,195
,245,125
1250 DATA -1,0,236,108,-1,1,158,114,-1,0
,106,140,-1,1,78,140,49,134,49,123
1260 DATA -1,0,250,137,-1,1,233,137
1360 DATA -1,2
READY.
```

Ecco qui due immagini dell'inchiostratore dello shuttle in azione; nella prima l'operazione è ancora in corso, mentre nella seconda è già terminata:

INCHIOSTRANDO LO SHUTTLE

DISEGNI CON SIN E COS

Alle pagg. 20 e 21 avete visto come i comandi SIN e COS del BASIC utilizzato dal Commodore possono produrre cerchi ed archi; questi non sono gli unici casi in cui potete applicare queste due funzioni. Date un'occhiata al primo programma di pag. 20, quello che costruiva un cerchio per punti: introducendo qualche piccola modifica potete ottenere dei risultati molto differenti.

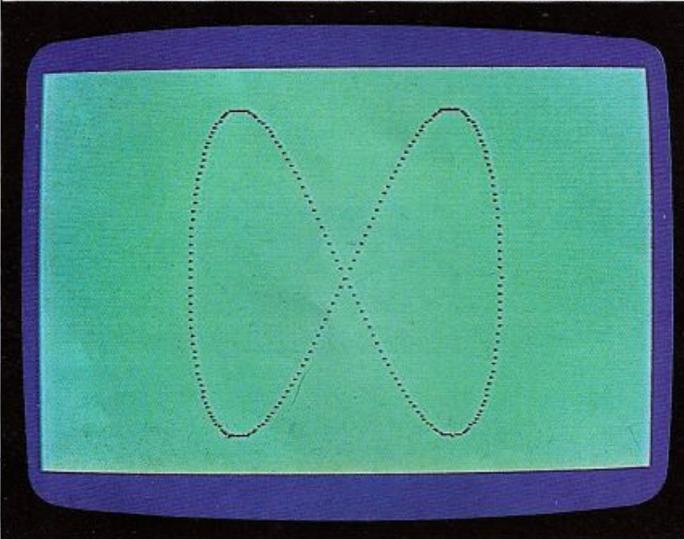
Per fare ciò avete ancora una volta bisogno delle otto subroutine per l'alta risoluzione perciò, se non sono già presenti in memoria, caricatele (LOAD) nel vostro computer prima di incominciare.

Come deformare un cerchio

Il programma "cerchio" di pag. 20 produce un cerchio perché le coordinate X ed Y variano in modo esattamente opposto: quando X è O, Y assume il suo valore massimo e viceversa. Cosa succederebbe, però, se voi li faceste deliberatamente variare in modo diverso? Provate a fare solo una modifica al programma cambiando l'argomento della funzione SIN alla riga 1030:

```
1030 LY=80*SIN(2*A)+100
```

Ecco il disegno che si otterrebbe (potete scegliere la combinazione di colori che preferite):

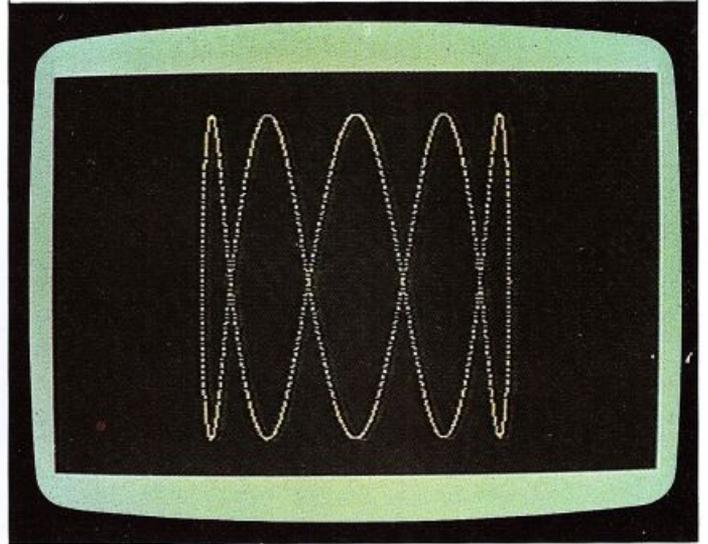
VISUALIZZAZIONE DI UNA FIGURA DI LISSAJOUS

Questa curva è detta "figura di Lissajous" dal nome del fisico francese che, per primo, la studiò.

Il numero di "onde" della figura dipende dal valore che moltiplica l'argomento di SIN e potete, quindi, adattare il programma per produrre un numero infinito di disegni come questo.

Ecco un altro modo di cambiarla:

```
1010 FOR A=0 TO 2*π/720
1030 LY=80*SIN(5*A)+100
```

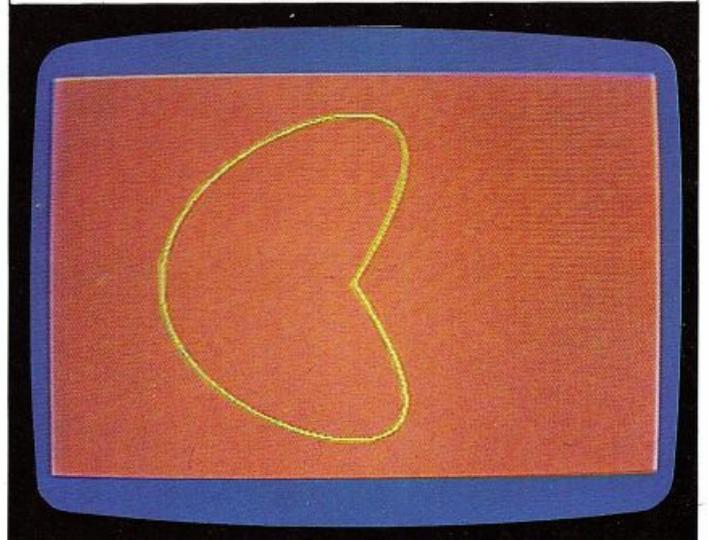
Questa volta l'angolo è stato moltiplicato per cinque, come potete notare guardando la figura:

VISUALIZZAZIONE DI UNA FIGURA DI LISSAJOUS

Come ottenere curve più complesse

Ora potete modificare il programma in modo differente. Provate ad immettere queste tre righe:

```
1010 R=0 : FOR A=0 TO 2*π STEP π/480
1020 LX=(R*COS(A)*R*SIN(0.5*A))+160
1030 LY=8*R*SIN(A)+100
```

Ecco cosa viene visualizzato se eseguite il programma dopo avere modificato i colori:

PROGRAMMA "FAGIOLO"

Il programma continuerà a disegnare se aumentate l'insieme dei valori dell'angolo; il prossimo programma raddoppia all'incirca l'estensione del ciclo FOR e, inoltre, aumenta la velocità di esecuzione tracciando delle brevi linee invece di singoli punti: ogni nuova posizione calcolata è il punto di partenza per la linea che dovrà essere tracciata alla prossima passata del ciclo FOR... NEXT:

PROGRAMMA "CLESSIDRA"

```
LIST
10 GOTO 1000
1000 COL=7 : POKE 53280,2 : GOSUB 100 :
GOSUB 200
1010 R=60
1020 X=0 : Y=0 : LX=160 : LY=100
1030 FOR A=0.2 TO 12.6 STEP 0.2
1040 I=R*COS(A)*SIN(0.5*A)
1050 J=R*SIN(A)
1060 NX=LX+I-X : NY=LY+J-Y
1070 GOSUB 600
1080 X=I : Y=J : NEXT A
1090 GOTO 1090
READY.
```

Potete fare degli esperimenti con questo programma per produrre una grande varietà di figure più complesse. Eccone uno che, però, è molto lento:

VISUALIZZAZIONE DI UNA SFERA

Il disegno è stato ottenuto cambiando i colori, e modificando le righe 1030 e 1040 nel seguente modo:

```
1030 FOR A=0.001 TO 1000 STEP 0.1
1040 I=R*COS(A)*SIN(0.98*A)
```

In questo caso SIN e COS operano su angoli che differiscono di pochissimo così, inizialmente, la figura risulta molto aperta ma, lentamente, il computer la colora fino a trasformarla nella sfera ricamata della schermata precedente. Il ricamo riprodotto e il modo con cui si sviluppa sono tanto interessanti quanto il disegno finale stesso. Potete, anche in questo caso, provare a modificare le due nuove righe per produrre figure differenti.

Come ottenere un grafico di SIN e COS

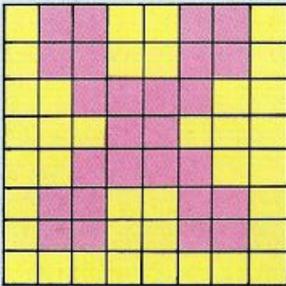
Come esempio finale dei disegni ottenuti mediante SIN e COS, ecco un programma che vi permette di confrontare come variano effettivamente i valori di SIN e COS per angoli compresi tra 0 e 2*π radianti (0 alla sinistra del grafico e 2*π alla destra). I valori di SIN e COS variano entrambi tra -1 e +1 ma, in questo programma, questa variazione è stata amplificata perché possiate osservarla più facilmente.

PROGRAMMA "SINUSOIDE E COSINUSOIDE"

```
LIST
10 GOTO 1000
1000 COL=151 : POKE 53280,8 : GOSUB 100
: GOSUB 200
1010 LX=3 : LY=95
1020 FOR A=π/36 TO 2*π STEP π/36
1030 NX=INT(A*50)
1040 NY=INT(95-80*SIN(A))
1050 GOSUB 600
1060 NEXT A
1070 LX=3 : LY=15
1080 FOR A=π/36 TO 2*π STEP π/36
1090 NX=INT(A*50)
1100 NY=INT(95-80*COS(A))
1110 GOSUB 600
1120 NEXT A
1130 GOTO 1130
READY.
```

COME INVENTARE NUOVI CARATTERI

Se osservate attentamente lo schermo, potrete riconoscere i singoli pixel che costituiscono ogni simbolo o carattere: ognuno di essi è costruito visualizzando una differente disposizione di pixel all'interno di una griglia 8x8. La lettera X, per esempio, è visualizzata in questo modo:

UN SINGOLO CARATTERE

Questa disposizione di pixel è memorizzata nello ROM, cosicché il carattere è disponibile non appena accendete il Commodore. All'interno di questa ROM (il "generatore di caratteri") ogni pixel di un carattere è rappresentato da un singolo bit di memoria e, quindi, una griglia 8x8 richiede otto byte di memoria e l'intero insieme dei caratteri (256 elementi) occupa 2K (2048 byte). Con il Commodore voi potete inventare dei caratteri personalizzati, sostituendo l'intero insieme di caratteri predefiniti, se lo desiderate.

Come modificare la struttura di un carattere

Dato che tutte le combinazioni di pixel sono memorizzate in ROM, viene istintivo chiedersi come sia possibile modificarle. Quello che dovete fare è ottenere che il Commodore ricerchi la struttura a pixel di un carattere in una differente area di memoria: se, per esempio, dite al vostro computer di estrarre i caratteri dalla RAM invece che dalla ROM, potete allora inserirvi dei caratteri personalizzati. Questa operazione è piuttosto semplice ma, una volta portata a termine, il vostro Commodore dimenticherà tutti i caratteri predefiniti. Se volete conservarne qualcuno dovete, perciò, copiare i caratteri che vi interessano (cifre e lettere, per esempio) dalla ROM alla RAM prima di cambiare zona di memoria. Il programma seguente copia 64 caratteri (le cifre e le lettere, appunto) dalla ROM alla RAM e quindi imposta il computer per l'uso dei caratteri in RAM. Noterete certamente che i caratteri in RAM sono memorizzati dalla locazione 2048 in poi; questo numero dovrebbe ormai esservi diventato familiare: è la locazione che il BASIC utilizza normalmente per memorizzare i programmi, perciò dovrete modificare l'area di memorizzazione BASIC come descritto a pag. 14, prima di eseguire il programma.

Siccome il computer deve trasferire un totale di 512 byte nelle nuove locazioni di memoria, ci vorranno alcuni secondi prima che l'operazione sia completata e i caratteri prescelti contenuti in ROM, siano memorizzati nell'area di RAM specificata:

COME SELEZIONARE I CARATTERI IN RAM

Quando eseguirete questo programma, dovreste essere subito in grado di determinare se esso funziona o meno, perché il normale carattere di cursore, che non è copiato dalla ROM alla RAM, dovrà essere sostituito da un rettangolo di pixel lampeggianti.

Costruite e memorizzate i vostri simboli personali

Supponiamo che vogliate visualizzare un piccolo simbolo sullo schermo: un razzo per un video gioco. La prima cosa che dovete fare è disegnare il carattere; si tratta di una tecnica simile a quella utilizzata per gli sprite, ma è più semplice.

DISEGNO DI UN SINGOLO CARATTERE

Al solito un pixel può essere "acceso" o "spento": i pixel accesi sono indicati in figura dai quadratini colorati. Per convertire tutto ciò in una serie di byte da memorizzare dovete considerare, per ogni singola riga, i valori dei bit accesi e sommarli in modo da ottenere un singolo byte. Seguendo questa procedura, ogni singolo carattere è convertito in 8 byte che, devono essere immessi (POKE) nella memoria alla fine dei dati relativi ai caratteri già esistenti. I caratteri più adatti a essere ridefiniti sono quelli ottenibili premendo anche il tasto di SHIFT, da SHIFT A a SHIFT Z.

Per aggiungere il disegno del razzo all'insieme dei caratteri, immettete le seguenti righe alla fine del programma precedente:

PROGRAMMA "RAZZO"

```
LIST

80 PRINT CHR$(147)
90 PRINT "        ";CHR$(97)
100 FOR T=1 TO 1000 : NEXT T
110 FOR C=0 TO 7
120 READ BYTE
130 POKE C+2568,BYTE
140 NEXT C
150 DATA 8,28,42,93,28,28,62,127
READY.
```

Quando eseguirete questo programma vedrete che il carattere che solitamente corrisponde a CHR$(97), apparirà sullo schermo ma, dopo brevissimo tempo, verrà sostituito da quello che avete appena immesso: il razzo.

Unire più caratteri personalizzati tra loro
La prima cosa che noterete sarà il fatto che il razzo è molto piccolo, in quanto esso occupa appena lo stesso spazio del video di una singola lettera. Benché i caratteri personalizzati siano basati su una griglia 8x8, non c'è ragione per cui i vostri simboli non debbano essere più grandi di un carattere, in modo da assegnare loro qualunque dimensione o forma vogliate. Ecco un disegno più complesso:

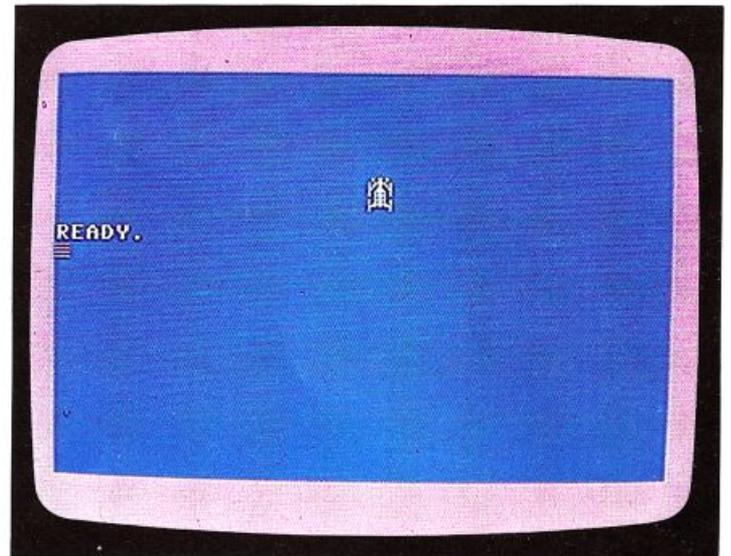
DISEGNO A PIÙ CARATTERI

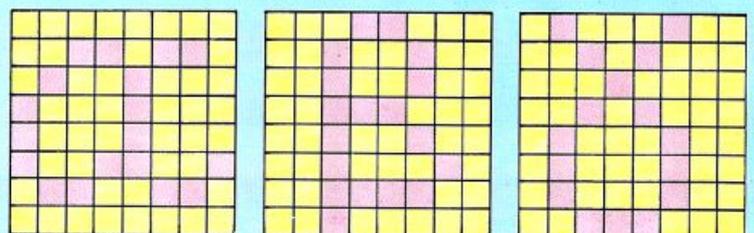
Valori delle colonne: 128 64 32 16 8 4 2 1 | 128 64 32 16 8 4 2 1

Byte risultante	Byte risultante
9	144
9	144
39	228
45	180
41	148
51	204
39	228
37	164
39	228
5	60
39	228
37	164
47	244
61	188
32	4
32	4

Nel programma seguente questi quattro caratteri sono memorizzati da CHR$(97) a CHR$(100). Ecco il programma completo e l'immagine che esso produce:

PROGRAMMA "QUATTRO CARATTERI"

```
LIST

200 PRINT CHR$(147) : POKE 53280,4 : POK
E 53281,6
210 FOR C=0 TO 31
220 READ BYTE : POKE C+2568,BYTE
230 NEXT C
240 X=20 : Y=5
250 POKE 214,Y : PRINT : POKE 211,X
260 PRINT CHR$(97);CHR$(98)
270 POKE 214,Y+1 : PRINT : POKE 211,X
280 PRINT CHR$(99);CHR$(100)
300 DATA 4,4,19,22,20,25,19,18,200
310 DATA 200,242,218,202,230,242,210
320 DATA 19,2,19,18,23,30,16,16
330 DATA 242,208,242,210,250,222,2,2
READY.
```

Questo programma suppone che i caratteri siano già stati ricopiati dalla ROM alla RAM ed è, inoltre, scritto in modo che il razzo possa essere visualizzato in un punto qualsiasi dello schermo modificando il valore di X e di Y alla riga 240: questi nuovi valori controllano la posizione in cui verrà visualizzato il carattere CHR$(97), la parte superiore sinistra del razzo.
Dal momento che esiste lo spazio per ridefinire un insieme completo di 256 caratteri, se lo volete, potete costruirvi ben più che un alfabeto completamente nuovo ed avere ancora abbastanza memoria per immagazzinare le lettere e i simboli predefiniti dal Commodore.

LETTERE PERSONALIZZATE

COME COSTRUIRE SPRITE SOFISTICATI

Uno sprite standard è costituito da una griglia di 24x21 pixel che può essere moltiplicato per un fattore 2 sia in orizzontale che in verticale. Questo significa che uno sprite esteso occupa una griglia di 48x42 pixel, cioè un'area quadrupla; questo aumento delle dimensioni dello sprite non significa che dovrete aggiungere altre istruzioni DATA al vostro programma: i registri V+23 e V+29 si occupano di tutto.
V+29 controlla l'espansione orizzontale mentre V+23 controlla l'espansione verticale; ognuno dei bit di questi due byte controlla un diverso sprite, in modo che per espandere lo sprite 0, dovrete porre a livello logico "1" il bit 0 di entrambi i registri mentre, per espandere lo sprite 1, dovrete compiere la medesima operazione sul bit 1 e così via. Lo sprite 0, dunque, verrà espanso in entrambe le direzioni per mezzo della seguente istruzione:

```
POKE V+23,1 : POKE V+29,1
```

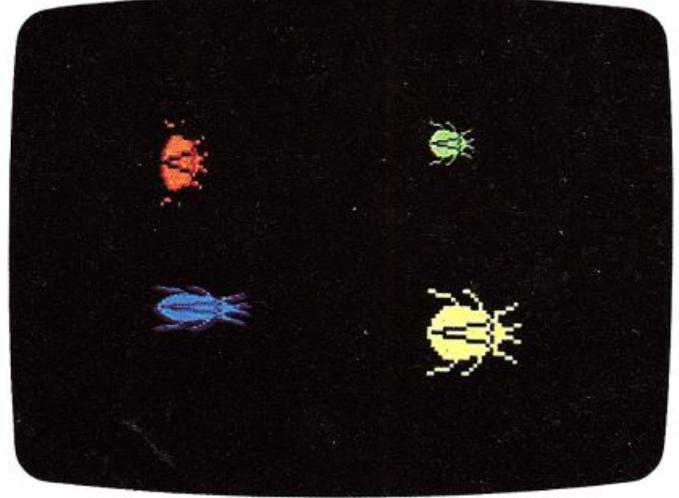
Per espandere lo sprite 1 dovrete "alzare" il bit 1: a questo bit è associato il valore 2 così, per espandere questo sprite, dovrete caricare (POKE) il valore 2 in entrambi i registri. Per espandere più sprite dovrete semplicemente sommare i valori da caricare: l'istruzione POKE V+23,3 espanderà verticalmente gli sprite 0 e 1.
Quando usate uno qualsiasi di questi registri di espansione, viene modificata solo la dimensione di ogni pixel; questo significa che, espandendo uno sprite, non otterrete nessun aumento della risoluzione del disegno: gli sprite saranno più grandi ma anche più "rozzi".
Il programma seguente permette di comparare tra loro tutte le possibili espansioni; esso memorizza gli sprite nell'area di memoria normalmente usata dal BASIC perciò, prima di immetterla, dovrete modificare l'area di memoria BASIC utilizzando il metodo di pag. 14.
Il programma costruisce gli sprite 0, 1, 2 e 3 a partire dagli stesssi valori (DATA) e quindi visualizza ciascuno di essi in ognuna delle quattro dimensioni possibili. La prima schermata è la sezione di controllo degli sprite:

PROGRAMMA DI ESPANSIONE DEGLI SPRITE

```
LIST -180

10 PRINT CHR$(147)
20 V=53248
30 POKE 53280,0 : POKE 53281,0
40 FOR C=0 TO 62
50 READ BYTE : POKE 2048+C,BYTE
60 NEXT C
70 POKE 2040,32 : POKE 2041,32
80 POKE 2042,32 : POKE 2043,32
90 POKE V+39,2
100 POKE V+40,5
110 POKE V+41,6
120 POKE V+42,7
130 POKE V,60 : POKE V+1,60
140 POKE V+2,200 : POKE V+3,60
150 POKE V+4,60 : POKE V+5,140
160 POKE V+6,200 : POKE V+7,140
170 POKE V+23,9
180 POKE V+29,12

READY.
```

La seconda schermata mostra i valori utilizzati per costruire ciascuno sprite:

PROGRAMMA DI ESPANSIONE DEGLI SPRITE

```
LIST 190-

190 POKE V+21,15
200 GOTO 200
400 DATA 192,96,0,60,16,0,2,8,0
410 DATA 1,4,2,15,228,12,31,254,112
420 DATA 63,255,64,127,254,192,127,193,9
9
430 DATA 240,63,124,135,254,240,240,63,1
24
440 DATA 127,193,99,127,254,192,63,255,6
4
450 DATA 31,254,112,15,228,12,1,4,2
460 DATA 2,8,0,60,16,0,192,96,0,0
READY.
```

VALORE DEI BIT PER GLI SPRITE A PIÙ COLORI

Ad ogni coppia di pixel di una riga della griglia corrispondente a uno sprite, è assegnato un unico valore. Usando i valori riportati nella tabella potete assegnare ad ogni coppia un colore qualsiasi su quattro, controllati da quattro registri differenti.

Valore della coppia di bit	Registro che controlla il colore della coppia di bit
00	Registro del colore dello schermo (53281)
01	Registro 1 per sprite a più colori (V+37)
10	Registri per normali sprite a colori (V+39–V+46)
11	Registro 2 per sprite a più colori (V+38)

DISEGNO DI SPRITE A PIÙ COLORI

Ecco disegnati due sprite a più colori di cui sono stati calcolati anche i byte per le istruzioni DATA relative. Il valore di ogni coppia di bit è selezionato in base alla tabella riportata alla pagina precedente.

0	64	0
0	16	0
0	16	0
9	85	128
38	102	96
42	102	160
174	170	232
170	234	168
170	171	168
186	186	168
171	171	184
170	186	168
174	170	232
170	238	168
186	170	184
42	187	160
43	170	160
10	171	128
10	234	128
2	170	0
0	168	0

0	5	64
1	85	64
1	84	0
0	20	0
0	20	0
0	20	0
0	17	0
0	17	0
0	17	0
0	17	0
0	64	64
0	64	64
0	66	96
1	10	168
9	138	184
42	170	184
42	234	168
42	234	168
42	162	160
42	160	128
10	128	0

Utilizzare gli sprite a più colori
In aggiunta alla possibilità di espandere gli sprite, esiste anche l'opportunità di costruirne utilizzando più di un colore. Ecco un programma che visualizza gli sprite precedentemente mostrati:

PROGRAMMA "SPRITE A PIÙ COLORI"

```
LIST -180

10 PRINT CHR$(147)
20 POKE 53280,0 : POKE 53281,0
30 V=53248
40 FOR C=0 TO 127
50 READ BYTE : POKE 2048+C,BYTE
60 NEXT C
70 Y1=0 : Y2=0
80 D1=1 : D2=1.1
90 POKE V+23,3 : POKE V+29,3 : POKE 2040
,32
100 POKE 2041,33
110 POKE V+39,2
120 POKE V+40,2
130 POKE V+37,5
140 POKE V+38,6
150 POKE V+28,3
160 POKE V+21,3
170 POKE V,100 : POKE V+1,Y1
180 POKE V+2,150 : POKE V+3,Y2

READY.
```

PROGRAMMA "SPRITE A PIÙ COLORI"

```
190 IF Y1>250 THEN Y1=0
200 IF Y2>250 THEN Y2=0
210 Y1=Y1+D1 : Y2=Y2+D2
220 GOTO 170
300 DATA 0,64,0,0,16,0
310 DATA 0,16,0,9,85,128
320 DATA 38,102,96,42,102,160
330 DATA 174,170,232,170,234,168
340 DATA 170,171,168,186,186,168
350 DATA 171,171,184,170,186,168
360 DATA 174,170,232,170,238,168
370 DATA 186,170,184,42,187,160
380 DATA 43,170,160,10,171,128
390 DATA 10,234,128,2,170,0
400 DATA 0,168,0,0
410 DATA 0,5,64,1,85,64,1,84,0,0,20,0
420 DATA 0,20,0,0,20,0,0,17,0,0,17,0
430 DATA 0,17,0,0,17,0,0,64,64,0,64,64
440 DATA 0,66,96,1,10,168,9,138,184
450 DATA 42,170,184,42,234,168
460 DATA 42,234,168,42,162,160
470 DATA 42,160,128,10,128,0,0
READY.
```

Come avete già visto, uno sprite colorato ha una risoluzione orizzontale di 24 pixel ma, se voi utilizzate più di un colore, questa risoluzione scende a 12 pixel. Questo non significa che lo sprite dimezza la sua larghezza, in quanto in uno sprite a più colori, ogni pixel viene visualizzato con una larghezza doppia.
In uno sprite normale ogni bit corrisponde ad un pixel sullo schermo mentre, in uno sprite a più colori, ogni coppia di bit della griglia determina il colore di una coppia di pixel sullo schermo.
Se una coppia di bit ha il valore binario 00, i pixel corrispondenti avranno lo stesso colore dello sfondo. Se la coppia assume il valore 10, il colore dei due pixel sarà quello memorizzato per lo sprite normale. La differenza si rivela quando una coppia di bit assume il valore 01 o 11 perché, allora, il colore dei pixel è preso da uno dei due speciali registri per i colori che si trovano alle locazioni V+37 e V+38 del chip VIC. Questo significa che, per uno sprite, possono essere utilizzati fino a quattro colori.
Ecco l'immagine prodotta dal programma "sprite a più colori":

IMMAGINE DEGLI SPRITE A PIÙ COLORI

ANIMARE GLI SPRITE

Quando create un programma che usa gli sprite, dovete decidere in quale zona della memoria del Commodore caricare i dati relativi a questi ultimi; come avete potuto constatare nel primo volume, nel buffer di input/output per la cassetta c'è spazio sufficiente per tre sprite a partire, rispettivamente, dalle locazioni 832, 896 e 960. Questa area di memoria può essere usata solo se non dovete impiegare contemporaneamente gli sprite e il registratore a cassette quando il programma è in esecuzione e, naturalmente, se non volete usare contemporaneamente più di tre sprite. In ogni caso esistono zone di memoria del Commodore in cui è possibile memorizzare molti più sprite: ne avete usata una, nelle due pagine precedenti, quando avete spostato l'area di memoria BASIC dalla sua normale allocazione.

Dove memorizzare gli sprite

Rendendo disponibile una grossa area di memoria per caricarvi i vostri sprite, potete memorizzare informazioni sufficienti per 32 di essi (quattro volte il numero di sprite che potete visualizzare contemporaneamente) dalla locazione 2048 in poi; risulta chiaro, allora, che potete avere a disposizione un grosso numero di figure differenti, le quali possono essere richiamate sullo schermo velocemente e con risultati molto interessanti.

Per vedere ciò che potete ottenere, ecco il disegno di un unico sprite e un programma che carica i dati relativi nell'area di memoria in cui si trova solitamente il BASIC; quando avete capito come funziona, potete provarne uno che modifica lo sprite per animarlo:

IL DISEGNO DI UN GRANCHIO

Valore dei bit

128 64 32 16 8 4 2 1 | 128 64 32 16 8 4 2 1 | 128 64 32 16 8 4 2 1

Valore per DATA

3	231	192
15	0	240
30	102	120
31	195	248
63	0	252
62	126	124
108	153	54
97	255	134
99	255	198
55	255	236
31	255	248
15	255	240
63	255	252
127	255	254
143	255	241
31	255	248
103	255	230
139	255	209
50	126	76
34	0	68
3	0	192

Poiché il programma usa lo spazio di memoria normalmente occupato dal BASIC, ricordatevi di spostare l'area di memoria BASIC prima di immetterlo (se lo avete dimenticato, rileggete la pag. 14). Quando eseguirete il programma, vedrete che lo sprite si muove lungo lo schermo; questo movimento è stato ottenuto sommando 2 alle coordinate X e Y in ogni passata del ciclo:

PROGRAMMA "GRANCHIO"

```
LIST -180

10 PRINT CHR$(147) : V=53248
20 FOR C=12 TO 62 : READ BYTE
30 POKE 2048+C,BYTE : POKE 2112+C,BYTE
40 POKE 2176+C,BYTE : NEXT C
50 FOR C=0 TO 2 : FOR K=0 TO 11
60 READ BYTE : POKE 2048+64*C+K,BYTE
70 NEXT K : NEXT C : POKE 53280,0 : POKE
 53281,0
80 POKE V+39,7 : POKE V+21,1
90 X=100 : DX=2 : S=32 : DS=-0.5
100 T=INT(X/256) : POKE 2040,S : POKE V+
23,3 : POKE V+29,3
110 POKE V+16,T
120 POKE V,X-T*256
130 POKE V+1,80
140 POKE 2040,INT(S)
150 IF S=32 OR S=34.5 THEN DS=-DS
160 IF X<50 OR X>280 THEN DX=-DX
170 X=X+DX : S=S+DS
180 GOTO 100

READY.
```

Come modificare la forma degli sprite

Ora è possibile compiere alcune operazioni sullo sprite stesso come, ad esempio, animarlo: si può ottenere questo effetto preparando in memoria due o più immagini dello stesso sprite, ma in differenti posizioni. Le locazioni dalla 2040 alla 2047 dicono al chip VIC del Commodore in che zona della memoria esso può trovare i dati relativi agli sprite da 0 a 7; tutto quello che dovete fare per animarne uno è far commutare rapidamente il chip VIC su una serie di immagini: è proprio come costruire un cartone animato utilizzando i numeri invece di carta e matita.
È necessario, dunque, caricare le locazioni da 2040 a 2047 con una serie di indirizzi che puntano ai dati relativi ad una sequenza di immagini differenti dello stesso sprite.
Ecco un esempio che mostra come si può fare. Continuando con il disegno del granchio potete costruire alcune altre versioni: i disegni qui sotto rappresentano il granchio con le chele in posizioni differenti. Dopo aver dato un'occhiata a questi disegni potete immettere il programma che utilizza i dati relativi per animare il granchio.
La cosa da notare in questo caso è che il programma non contiene l'insieme completo dei dati relativi alle tre versioni: la maggior parte di essi è identica in tutti i disegni, e memorizzarli ogni volta farebbe sprecare molta memoria. Solo le quattro righe superiori di ogni versione sono memorizzate separatamente.

DIFFERENTI VERSIONI DEL GRANCHIO

Valori per DATA

3	224	0
15	7	224
30	103	240
31	193	248
63	0	252
62	126	124
108	153	54
97	255	134
99	255	198
55	255	236
31	255	248
15	255	240
63	255	252
127	255	254
143	255	241
31	255	248
103	255	230
139	255	209
50	126	76
34	0	68
3	0	192

Valori per DATA

0	7	192
7	224	240
15	230	120
31	131	248
63	0	252
62	126	124
108	153	54
97	255	134
99	255	198
55	255	236
31	255	248
15	255	240
63	255	252
127	255	254
143	255	241
31	255	248
103	255	230
139	255	209
50	126	76
34	0	68
3	0	192

PROGRAMMA "GRANCHIO ANIMATO"

```
LIST -180

10 PRINT CHR$(147)
20 V=53248
30 POKE 53280,6 : POKE 53281,6
40 FOR C=0 TO 62
50 READ BYTE : POKE 2048+C,BYTE
60 NEXT C
70 X=100 : Y=100
80 DX=2 : DY=2
90 POKE 2040,32 : POKE V+23,1 : POKE V+2
9,1
100 POKE V+39,7 : POKE V+21,1
110 T=INT(X/256)
120 POKE V+16,T
130 POKE V,X-T*256
140 POKE V+1,Y
150 IF X<60 OR X>280 THEN DX=-DX
160 IF Y<60 OR Y>180 THEN DY=-DY
170 X=X+DX : Y=Y+DY
180 GOTO 110

READY.
```

```
LIST 190-

200 DATA 63,0,252,62,126,124
210 DATA 108,153,54,97,255,134
220 DATA 99,255,198,55,255,236
230 DATA 31,255,248,15,255,240
240 DATA 63,255,252,127,255,254
250 DATA 143,255,241,31,255,248
260 DATA 103,255,230,139,255,209
270 DATA 50,126,76,34,0,68
280 DATA 3,0,192
290 DATA 3,224,0,15,7,224
300 DATA 30,103,240,31,193,248
310 DATA 3,231,192,15,0,240
320 DATA 30,102,120,31,195,248
330 DATA 0,7,192,7,224,240
340 DATA 15,230,120,31,131,248
READY.
```

Ora, mentre il granchio si muove, le sue chele si aprono e si chiudono come in un cartone animato, come potete notare in questi primi piani sul programma in esecuzione.

VISUALIZZAZIONE DEL GRANCHIO ANIMATO

SOVRAPPOSIZIONI E SCONTRI

Cosa succede quando uno sprite si muove in una posizione dello schermo già occupata da qualcos'altro? Se avete eseguito un programma che utilizza gli sprite senza prima cancellare lo schermo, avrete probabilmente scoperto che essi sembravano "fluttuare" sopra il testo precedente, ricoprendolo ma non cancellandolo. D'altra parte, se due sprite si scontravano, il risultato poteva essere molto differente.

Predisporre sovrapposizioni con priorità tra gli sprite

Voi potete specificare al vostro Commodore come sovrapporre sprite e sfondo e quali operazioni compiere nel caso di uno scontro tra sprite.

Quando due sprite si scontrano, esiste una semplice regola per determinare quale dei due apparirà "sopra" e quale "sotto": lo sprite col numero più basso verrà sempre visualizzato al di sopra di uno con un numero più alto.

Questa regola fornisce, dunque, la priorità più alta allo sprite 0 e la più bassa al numero 7: se, per esempio, lo sprite 2 si scontra con lo sprite 4, il primo sembrerà passare sopra al secondo.

La priorità tra sprite, dunque, si risolve semplicemente in una attenta numerazione degli stessi.

Quando uno sprite si scontra, con un oggetto appartenente allo sfondo, la situazione è leggermente diversa. La locazione di memoria 53275 (V+27 del chip VIC) controlla la priorità tra sprite e sfondo; ogni bit stabilisce la priorità per uno sprite; il bit 0 controlla lo sprite 0, il bit 1 lo sprite 1 e così via. Se portate a livello "1" un bit in questo byte allora, in uno scontro, lo sprite corrispondente passerà dietro lo sfondo, mentre, portando un bit a livello "0", il suo sprite sembra passare davanti allo sfondo.

Potete usare questa tecnica per realizzare alcuni interessanti effetti. Il programma seguente mostra la priorità tra sprite al lavoro; esso crea una situazione che è impossibile nella realtà, ma molto facile da realizzare con gli sprite:

PROGRAMMA "PRIORITÀ TRA SPRITE"

```
LIST -200

10 PRINT CHR$(147);CHR$(158)
20 POKE 53280,5 : POKE 53281,6
30 V=53248
40 FOR C=0 TO 62
50 READ BYTE : POKE 2048+C,BYTE
60 NEXT C
70 POKE 2040,32 : POKE 2041,32
80 POKE V+27,1
90 POKE V+39,8 : POKE V+40,4
100 POKE 214,10 : PRINT
110 FOR X=5 TO 30
120 POKE 211,X : PRINT "●";
130 NEXT X
140 POKE V+21,3:POKE V+23,3:POKE V+29,3
150 Y1=60 : Y2=200
160 D1=6 : D2=-6
170 POKE V,150 : POKE V+2,165
180 POKE V+1,Y1 : POKE V+3,Y2
190 IF Y1<60 OR Y1>200 THEN D1=-D1
200 IF Y2<60 OR Y2>200 THEN D2=-D2

READY.
```

IMMAGINI DEL PROGRAMMA "PRIORITÀ TRA SPRITE"

In questo programma ci sono due sprite, il numero uno e il numero 0: lo sprite 0 ha la priorità. Ciò che rende in-

teressante il programma è l'ordine con cui sono state impostate le priorità tra sprite e sfondo: lo sprite 1 ha la priorità rispetto allo sfondo, mentre il numero 0 si muove sotto di esso, creando così uno strano effetto sullo schermo.

Rilevare gli scontri sullo schermo

Un altro importante aiuto fornito dal Commodore per l'animazione degli sprite è la rilevazione degli scontri. Il computer suppone che sia avvenuto uno scontro se un qualsiasi pixel acceso in uno sprite animato si trova nella medesima posizione di un pixel acceso di un qualsiasi altro oggetto sullo schermo; ciò significa che, nei programmi che usano missili e bersagli, per esempio, non importa quanto siano complessi il missile e il bersaglio: le istruzioni da immettere sono sempre le stesse.

Ci sono due tipi di scontri possibili sullo schermo: uno in cui uno sprite urta un altro sprite e l'altro in cui uno sprite urta un oggetto appartenente allo sfondo; entrambe queste situazioni sono trattate nello stesso modo; cioè il computer "alza" i singoli bit dei registri del chip VIC per segnalare che uno scontro ha avuto luogo. Due registri VIC sono interessati da questa operazione, V+30 e V+31: il primo rileva gli scontri sprite/sprite mentre il secondo quelli tra sprite e sfondo. In tutti e due i registri sono significativi i singoli bit, i quali controllano gli sprite nel solito modo.

Per cominciare, il chip VIC azzera entrambi i registri e quindi, quando ha luogo uno scontro, i bit opportuni vengono portati a "1"; questo valore è memorizzato finché voi non effettuate una lettura del registro mediante l'istruzione PEEK, poi esso è azzerato nuovamente: questo significa che il vostro software ha la possibilità di scoprire ogni scontro, non importa quando esso sia avvenuto; in ogni caso, dovete ricordarvi che la lettura del registro lo azzera, perciò la cosa migliore da fare è memorizzare il suo contenuto in una variabile in modo da non perderlo prima di avere finito la sua elaborazione.

Il programma che segue è simile al precedente, tranne per il fatto che esso rileva e segnala gli scontri:

PROGRAMMA "SCONTRO TRA SPRITE"

```
10 S=54272 : POKE S+24,15
20 POKE S+1,115 : POKE S,88
30 POKE S+5,0 : POKE S+6,240
40 PRINT CHR$(147);CHR$(156)
50 POKE 53280,0 : POKE 53281,0
60 V=53248
70 FOR C=0 TO 62
80 READ BYTE : POKE 2048+C,BYTE
90 NEXT C
100 POKE 2040,32 : POKE 2041,32
110 POKE V+27,2 : POKE V+23,3 : POKE V+2
9,3
120 POKE V+39,3: POKE V+40,6
130 POKE 214,15 : PRINT
140 PRINT CHR$(18);"
                                       "
150 POKE V+21,3
160 X=120 : Y=130
170 D1=3 : D2=-1.5
180 POKE V,X+D1 : POKE V+2,170
190 POKE V+1,135 : POKE V+3,Y
200 POKE S+4,16 : POKE V+27,(D1+3)/2

READY.
```

PROGRAMMA "SCONTRO TRA SPRITE"

```
LIST 210-

210 IF X<80 OR X>200 THEN D1=-D1
220 IF Y<60 OR Y>135 THEN D2=-D2
230 X=X+D1 : Y=Y+D2
240 SS=PEEK(V+30) : SB=PEEK(V+31)
250 IF SS=0 AND SB=0 THEN 180
260 POKE S+4,17 : PRINT"*"; : GOTO 180
270 DATA 3,231,192,15,0,240
280 DATA 30,102,120,31,195,248
290 DATA 63,0,252,62,126,124
300 DATA 108,153,54,97,255,134
310 DATA 99,255,198,55,255,236
320 DATA 31,255,248,15,255,240
330 DATA 63,255,252,127,255,254
340 DATA 143,255,241,31,255,248
350 DATA 103,255,230,139,255,209
360 DATA 50,126,76,34,0,68
370 DATA 3,0,192
READY.
```

Ogni volta che due sprite si scontrano, si stampa un asterisco. Gli asterischi potrebbero far scorrere lo schermo verso l'alto creando così uno scontro ripetuto:

VISUALIZZAZIONE DI "SCONTRI TRA SPRITE"

TORTE E GRAFICI

La grafica del computer è l'ideale per visualizzare informazioni che potete rilevare con un'occhiata, e uno dei grafici più facilmente interpretabili che il Commodore può produrre è il grafico a "torta". I grafici "a torta" sono particolarmente adatti per mostrare i rapporti tra varie grandezze, o come elementi differenti costituiscano una parte dell'intera grandezza, evidenziando le diverse percentuali.

Impostare un grafico a torta

Per produrre un grafico "a torta", innanzitutto è necessario disegnare un cerchio, poi dovete tracciare le linee che dividono le varie "fette". Nel programma seguente è evidenziata solo una fetta corrispondente ad un quarto del cerchio. Per fare in modo che il Commodore produca un cerchio, dovete utilizzare l'alta risoluzione perciò, prima di eseguire questo programma, ricordatevi che è necessario spostare l'area di memoria BASIC (vedere la pag. 14) e quindi caricare l'insieme completo delle otto subroutine grafiche che dovreste aver registrato su cassetta o disco. Ecco il semplice programma per il grafico "a torta" che richiama questa subroutine:

PROGRAMMA "DIAGRAMMA A TORTA PREDEFINITO"

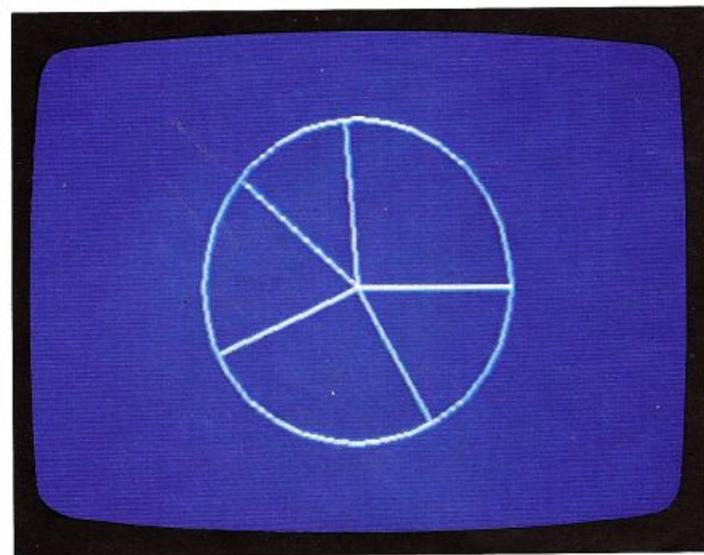
```
LIST
10 GOTO 1000
1000 POKE 53280,6 : COL=5
1010 GOSUB 100 : GOSUB 200
1020 XC=160 : YC=100
1030 RAD=80 : GOSUB 700
1040 LX=240 : LY=100
1050 NX=160 : NY=100 : GOSUB 600
1060 NX=160 : NY=20 : GOSUB 600
1070 GOTO 1070
READY.
```

Il programma disegna un cerchio al centro dello schermo (coordinate 160, 100). La riga 1050 traccia il primo raggio da destra verso il centro, mentre la riga 1060, poi, traccia il secondo raggio ad angolo retto per delineare la fetta.

Aggiungere altre "fette" alla torta

Potreste continuare da questo punto e aggiungere altre "fette" per completare il grafico, ma un programma di questo tipo non sarebbe molto utile: un programma che riproduce sempre il medesimo diagramma "a torta", fornirebbe sempre la stessa immagine. Ciò che serve è un programma che elabori i dati che voi immettete:

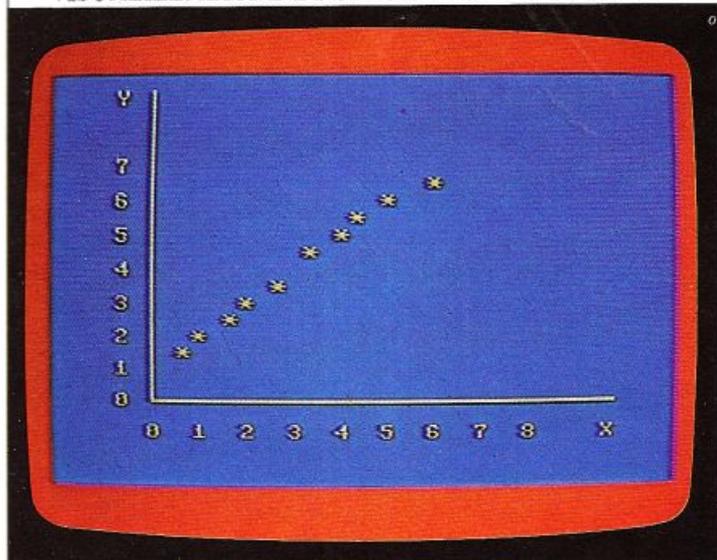
PROGRAMMA "DIAGRAMMA A TORTA VARIABILE"

```
LIST

10 GOTO 1000
1000 PRINT CHR$(147) : T=0
1010 INPUT "QUANTE FETTE";N : PRINT
1020 DIM P(N) : FOR C=1 TO N
1030 PRINT "AMPIEZZA DELL'ANGOLO";C;" ";

1040 INPUT S : T=T+S : P(C)=T : NEXT C
1050 P(0)=0 : POKE 53280,6 : COL=22
1060 GOSUB 100 : GOSUB 200
1070 XC=160 : YC=100
1080 RAD=80 : GOSUB 700
1090 FOR C=0 TO N-1
1100 LX=160 : LY=100 : S=P(C)*2*π/T
1110 NX=INT(160+(80*COS(S)+0.5))
1120 NY=INT(100+(80*SIN(S)+0.5))
1130 GOSUB 600 : NEXT C
1140 GOTO 1140
READY.
```

Nel programma le righe da 1000 a 1040 determinano quante fette vogliamo sul diagramma e quanto deve essere grossa ogni fetta: potete specificare direttamente questi valori. Alla riga 1020 è presente il comando DIM, che predispone un qualcosa detto "array" che è un modo per memorizzare e consultare informazioni

rapidamente. Gli array sono ampiamente descritti alle pagg. 52 e 53. Partendo dalle informazioni contenute nell'array, il programma si calcola l'angolo relativo ad ogni fetta; questo calcolo e il disegno vero e proprio, sono eseguiti dalle righe numerate da 1070 a 1130. Dal momento che il diagramma è disegnato in alta risoluzione, non potete visualizzare alcuna scritta su di esso.

Aggiungere informazioni ai grafici

I diagrammi "a torta" sono utili per mostrare in che modo è suddivisa una certa grandezza; gli altri grafici, d'altra parte, mostrano come sono correlati due insiemi dati. Ecco una semplice visualizzazione di un grafico fatto dal Commodore; esso è prodotto in bassa risoluzione (pagina di testo), cosicché è possibile aggiungervi scritte e crearlo più rapidamente:

VISUALIZZAZIONE DI UN GRAFICO PREDEFINITO

Non è necessario essere un matematico per ricavare utili informazioni da questo grafico: man mano che il tempo trascorre (sull'asse orizzontale), la quantità riportata sull'asse verticale aumenta. Il programma che predispone questo grafico deve disegnare gli assi, le loro suddivisioni, e visualizzare gli asterischi.

PROGRAMMA "GRAFICO PREDEFINITO"

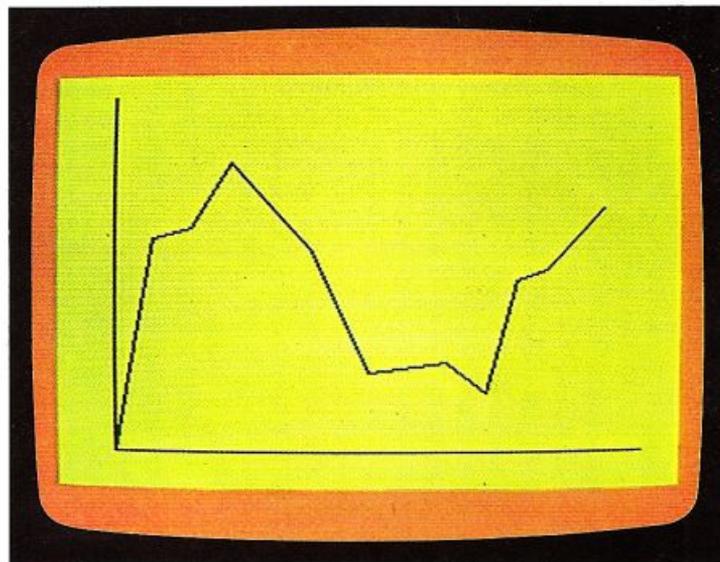
```
LIST

10 PRINT CHR$(147)
20 POKE 53280,2 : POKE 53281,6
30 PRINT CHR$(158) : FOR Y=0 TO 17
40 POKE 214,Y : PRINT : POKE 211,6
50 PRINT CHR$(125) : NEXT Y
60 PRINT TAB(6);CHR$(173);
70 FOR X=7 TO 35
80 PRINT TAB(X);CHR$(96); : NEXT X
90 PRINT : PRINT : PRINT "     0  1  2
 3  4  5  6  7  8    X" : PRINT
100 C=0
110 FOR Y=18 TO 4 STEP -2
120 POKE 214,Y : PRINT : POKE 211,3
130 PRINT C : C=C+1 : NEXT Y
140 POKE 214,0 : PRINT : PRINT "    Y"
150 FOR D=1 TO 10 : READ X,Y
160 POKE 214,Y : PRINT : POKE 211,X
170 PRINT "*" : NEXT D
180 GOTO 180
190 DATA 8,15,9,14,11,13,12,12,14,11
200 DATA 16,9,18,8,19,7,21,6,24,5
READY.
```

I due insiemi di informazioni sono contenuti nelle istruzioni DATA delle righe 190 e 200; le righe da 30 a 140 disegnano gli assi del grafico e le loro suddivisioni. Mentre il ciclo seguente legge 10 coppie di valori dalle istruzioni DATA e li usa come coordinate per visualizzare i punti del grafico.

Costruire un plotter grafico in alta risoluzione

La visualizzazione del grafico predefinito è piuttosto rozza, dal momento che esso usa la pagina in bassa risoluzione, la quale possiede solo 504 posizioni di carattere all'interno dell'area grafica di 28 colonne per 18 linee. Per ottenere un grafico più dettagliato, dovete passare in alta risoluzione e usare le subroutine grafiche. Il programma seguente compie queste operazioni, usando ancora le informazioni contenute nelle istruzioni DATA; esso è più lento ma l'immagine è più accurata:

PROGRAMMA "GRAFICO IN ALTA RISOLUZIONE"

```
LIST
10 GOTO 1000
1000 COL=7 : POKE 53280,8
1010 GOSUB 100 : GOSUB 200
1020 LX=30 : LY=10
1030 NX=30 : NY=180 : GOSUB 600
1040 NX=300 : NY=180 : GOSUB 600
1050 LX=30 : LY=180
1060 FOR C=1 TO 10 : READ NX,NY
1070 GOSUB 600 : NEXT C
1080 GOTO 1080
1090 DATA 50,75,70,70,90,40,130,80
1100 DATA 160,140,200,135,220,150
1110 DATA 235,95,250,90,280,60
READY.
```

Potete facilmente adattare questo programma in modo che, invece di utilizzare delle istruzioni di DATA al suo interno, esso accetti dei vostri valori di INPUT prima di disegnare il grafico, proprio come il programma per il diagramma "a torta"; in questo modo potete visualizzare sul grafico, in pochi secondi, qualsiasi informazione.

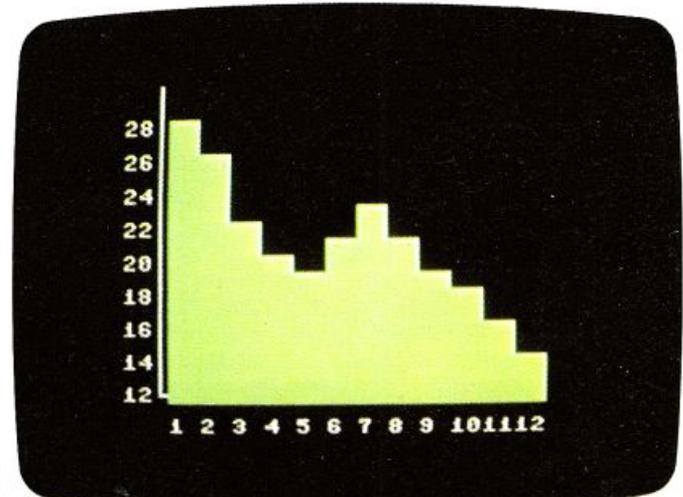
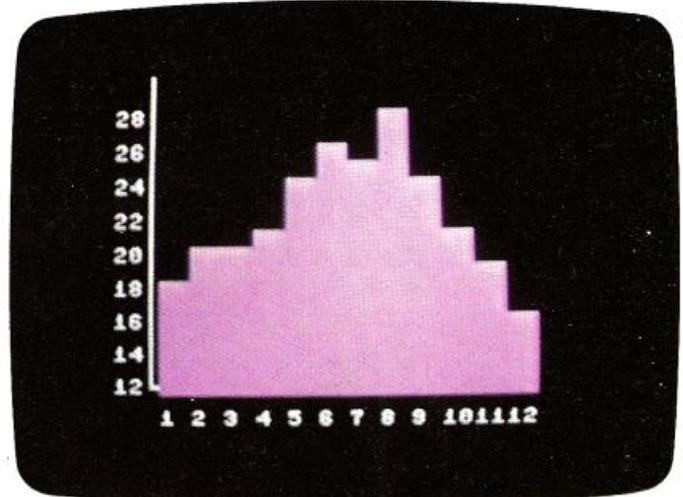
DIAGRAMMI A BARRE

Avendo visto come il Commodore può produrre diagrammi "a torta" e grafici in alta risoluzione, potete ora aggiungere un terzo modo di visualizzare le informazioni, usando diagrammi a barre in bassa risoluzione.
Nei diagrammi a barre un dato non è mostrato come un singolo punto, ma come una colonna la cui altezza dipende dal valore del termine corrispondente. Essi sono usati frequentemente per mostrare le variazioni dei cambi valutari, dei voti nelle elezioni e così via, e potete usare facilmente il vostro Commodore per visualizzare i vostri dati personali in un modo istantaneo e grafico.

Come scrivere un programma per diagrammi a barre
Dal momento che un diagramma a barre non è che un particolare tipo di grafico, potete usare la stessa tecnica di programmazione per produrne uno; la differenza principale è che invece di disegnare dei singoli punti alle coordinate fornite, il programma deve disegnare una colonnina. Con il Commodore è possibile ottenere barre molto semplicemente usando un simbolo grafico opportuno (CHR$(18)); potete poi colorarle per rendere più chiara l'interpretazione del grafico. Il programma seguente esegue tutto ciò usando dati che potete immettere da tastiera; dal momento che esso utilizza la bassa risoluzione, potete aggiungere, senza alcun problema, delle scritte al grafico:

PROGRAMMA "SEMPLICE DIAGRAMMA A BARRE"

L'asse Y (verticale) è tracciato dalle righe da 30 a 50 nella stessa posizione del primo grafico di pag. 35; l'istruzione PRINT alla riga 70, visualizza sull'asse X (orizzontale) i numeri da 1 a 12, i quali rappresentano i mesi dell'anno.
Il programma disegna barre che sono larghe due caratteri; se volete aumentare il massimo numero di barre visualizzabili in ogni grafico, potete ridurre la loro larghezza a un singolo carattere ma, se fate ciò, ricordatevi che dovete modificare la loro disposizione di stampa per fare in modo che le barre appaiano alla giusta distanza. Ecco in azione il programma per le barre a doppia larghezza visualizzate in due diversi colori:

VISUALIZZAZIONE DEI DIAGRAMMI A BARRE

Unire due diagrammi
Questi primi diagrammi possono mostrare solo una lista di valori, ma è possibile ristrutturarli in modo che essi possano visualizzare più di un insieme di informazioni. Potete, per esempio, voler vedere sia i valori massimi che quelli minimi di una temperatura sullo stesso diagramma: non dovete riscrivere l'intero programma, sono necessarie solo poche aggiunte:

```
105 FOR N=1 TO 2
115 IF N=2 THEN PRINT CHR$(158)
195 NEXT N
```

Tutto procederà come nel caso precedente finché non avete finito di immettere il primo gruppo di dati; il programma assegnerà quindi a N il valore 2 e disegnerà le colonne di colore giallo invece che magenta. Il secondo insieme di dati deve contenere generalmente valori minori rispetto al primo, altrimenti il diagramma magenta risulterà coperto interamente da quello giallo:

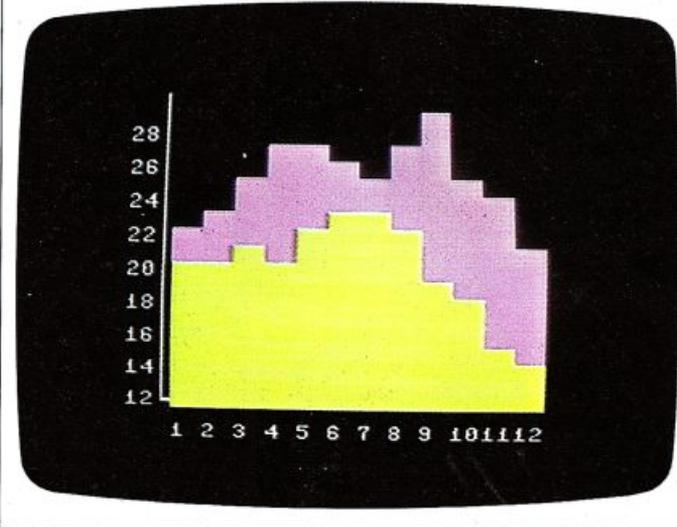
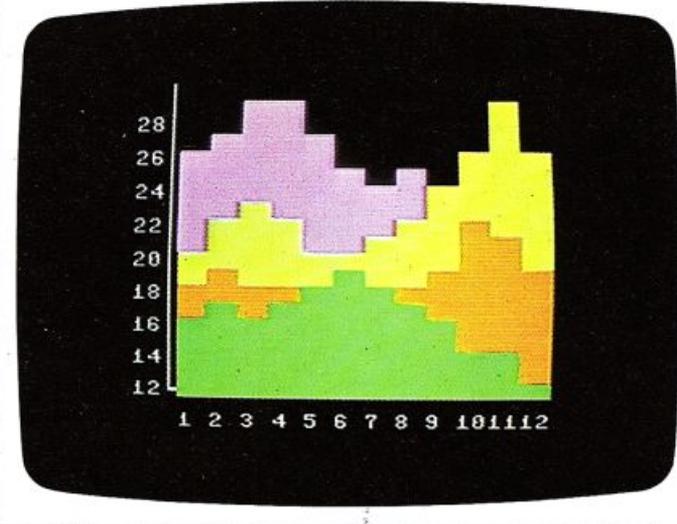
DIAGRAMMA A DUE COLORI

Potete estendere il procedimento a un numero qualsiasi di diagrammi sovrapposti, aumentando il limite superiore del ciclo FOR...NEXT alla riga 105 e aggiungendo righe tra la 110 e la 120 per cambiare colore. Ecco un diagramma con quattro gruppi di informazioni:

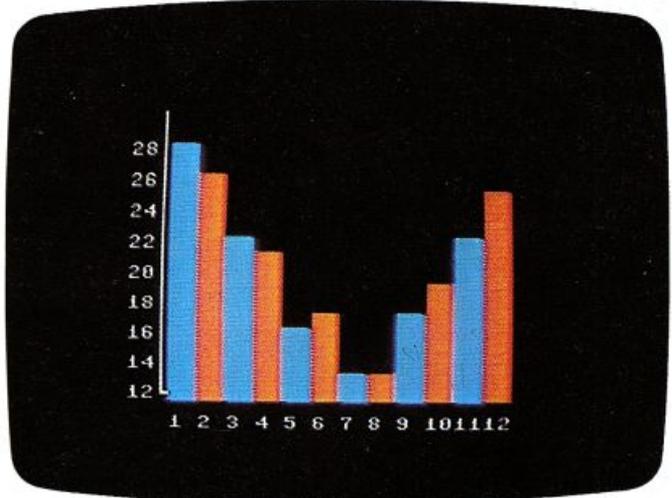
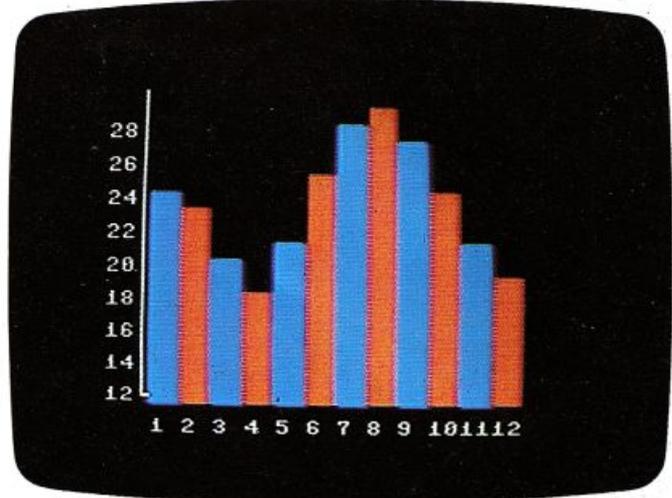
DIAGRAMMA A 4 COLORI

Come migliorare i diagrammi alternando i colori

Uno dei problemi che insorge con i diagrammi che utilizzano un solo colore per ogni insieme di dati, è che non si possono distinguere le singole barre, ed è difficile metterle in relazione alla scala riportata sull'asse orizzontale. Potete aggirare questo ostacolo utilizzando ancora due colori ma, questa volta, alternandoli man mano che vengono disegnate le singole barre; è allora facile determinare quale è la barra relativa ad un determinato valore dell'asse X.

Il programma seguente è un adattamento del "semplice programma a barre"; se eliminate le righe che gli permettono di visualizzare più di un insieme di dati, potete modificarlo per ottenere un diagramma a colori alternati.

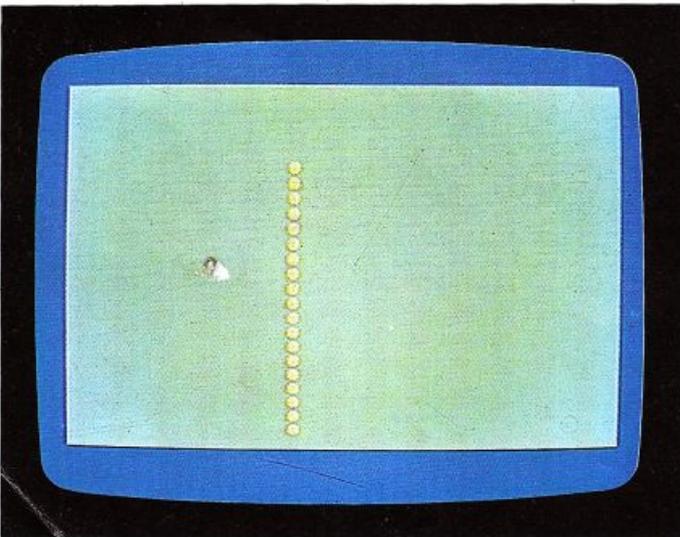
Invece di utilizzare un colore predefinito, quest'ultimo è ora controllato dalla variabile A; per determinare quando usare il colore rosso o blu, è stato utilizzato un ciclo FOR...NEXT unito ad una istruzione IF...THEN. Potete utilizzare questo ciclo di cambio colore con un numero qualsiasi di colori. Ecco il programma e alcuni dei diagrammi che esso produce:

DIAGRAMMI A COLORI ALTERNATI

```
10 PRINT CHR$(147);CHR$(5)
20 POKE 53280,0 : POKE 53281,0
30 FOR Y=0 TO 17
40 POKE 214,Y : PRINT : POKE 211,6
50 PRINT CHR$(125) : NEXT Y
60 PRINT TAB(6);CHR$(173)
70 PRINT : PRINT "       1 2 3 4 5 6 7 8
 9 101112"
80 C=12 : FOR Y=18 TO 2 STEP -2
90 POKE 214,Y : PRINT : POKE 211,3
100 PRINT C : C=C+2 : NEXT Y : M=1
110 FOR A=1 TO 2 : PRINT CHR$(5)
120 POKE 214,21 : PRINT : POKE 211,12
130 PRINT "X";M;"= "; : INPUT T
140 POKE 214,21 : PRINT : POKE 211,12
150 PRINT "                            "
160 PRINT CHR$(34-3*A)
170 FOR R=18 TO 30-T STEP -1
180 POKE 214,R : PRINT : POKE 211,M*2+5
190 PRINT CHR$(18);"  " : NEXT R
200 M=M+1 : NEXT A : IF M<12 THEN 110
210 GOTO 210
READY.
```

DISEGNI PRODOTTI DALLA FORZA DI GRAVITÀ

Alle pagg. 24 e 25 avete visto come il Commodore può produrre dei disegni "naturali", cioè figure che potete ritrovare in natura.
Per costruire queste figure potete cimentarvi semplicemente con le routine grafiche e vedere cosa succede; ma se volete che il vostro computer simuli in modo attendibile qualcosa che si muove, vi sarebbe di grande aiuto conoscere come quel qualcosa si muove nella realtà per poter programmare gli stessi movimenti sullo schermo del computer.

Come il Commodore lascia cadere una pallina
Alle pagg. 8 e 9 l'istruzione IF...THEN è stata usata per far rimbalzare una pallina che si muoveva in linea retta a velocità costante. Nella schermata sottostante è riportato un breve programma che dimostra come potete cominciare a simulare più realisticamente una pallina che cade (la schermata più in basso include anche le immagini della pallina che sono normalmente cancellate dalla prima istruzione della riga 80).

PROGRAMMA "CADUTA SEMPLICE"

Gli oggetti che cadono sono influenzati da numerosi fattori (la forza di gravità, la resistenza dell'aria, l'attrito superficiale e qualcosa detto "coefficiente di penetrazione") i quali li fanno muovere in maniera complessa; comunque non è necessario essere dei fisici per scrivere un programma più realistico del precedente. Se fate cadere una pallina, essa raggiunge il suolo e rimbalza indietro, e questo è tutto quello che dovete sapere per far rimbalzare una pallina sullo schermo.

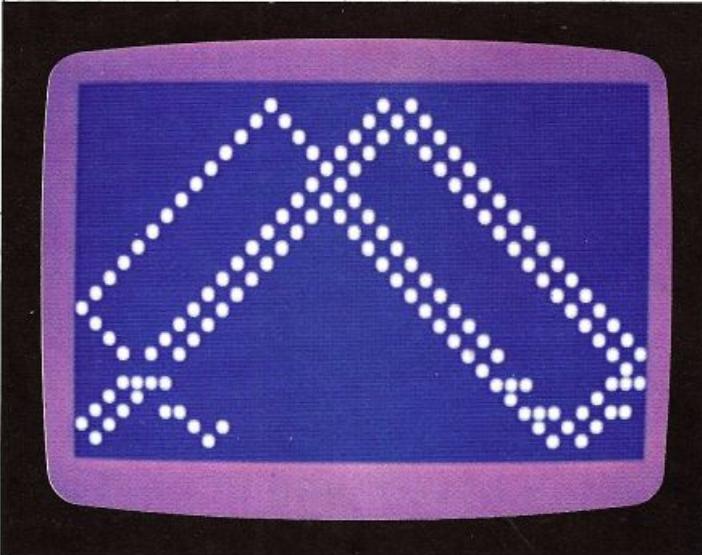
Programmare movimenti in due direzioni
Nel programma "caduta semplice", la riga 50 visualizza la pallina vicino alla parte superiore dello schermo, e dopo una pausa di 2 secondi, essa comincia a muoversi verso il basso. La riga 80 la cancella; la sua coordinata verticale è incrementata di 1, e infine, la pallina è visualizzata di nuovo. Se eseguite il programma, scoprirete che, benché la pallina stia indiscutibilmente cadendo verso il basso, il suo movimento non appare molto realistico; inoltre il programma termina brutalmente quando la pallina raggiunge l'ultima linea dello schermo. Il programma seguente migliora il risultato in modo considerevole, facendo in modo che la pallina si muova anche lateralmente:

PROGRAMMA "CADUTA OBLIQUA"

La variabile H rappresenta il valore da sommare alla posizione orizzontale e V quello da sommare alla posizione verticale: ad ogni ciclo, V è sommata al numero di linea e H a quello di colonna. Ora è semplice modificare il movimento in una direzione qualsiasi: potete, per esempio, far rimbalzare la palla aggiungendo:

```
85 IF C<1 OR C>38 THEN H=-H
86 IF R<1 OR R>22 THEN V=-V
```

Avete già visto questa tecnica in precedenza usando gli operatori AND e OR con IF...THEN, perciò queste due righe non dovrebbero presentare particolari problemi. Se eliminate le righe che cancellano la pallina quando si muove, potrete vedere un'immagine come questa:

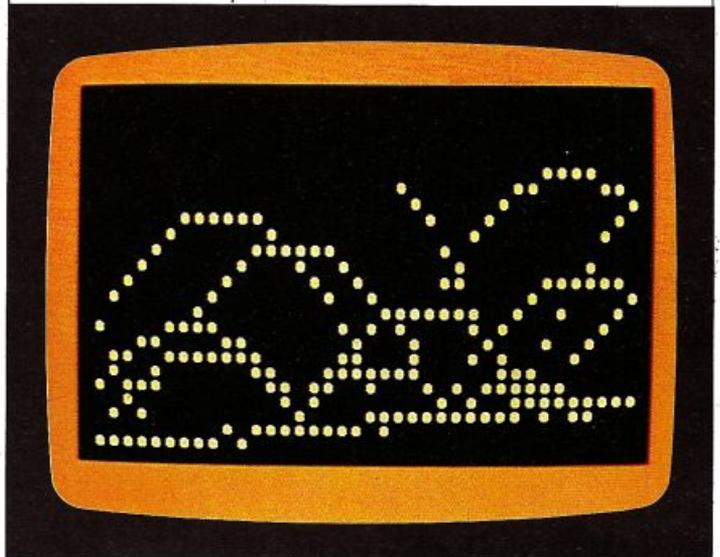
VISUALIZZAZIONE DELLA CADUTA OBLIQUA

Una gravità controllata dal computer

Benché la palla rimbalzi lungo lo schermo, essa non appare del tutto realistica. La ragione di ciò è che essa non è soggetta a nessuna forza di gravità; potete però aggiungere voi una "forza" simile alla gravità, la quale attira la pallina in una direzione qualsiasi o che cambia direzione durante l'esecuzione del programma. La gravità agisce verso il basso così, quando la pallina si muove dalla parte superiore a quella inferiore dello schermo, essa dovrebbe accelerare mentre, quando rimbalza indietro dovrebbe rallentare fino ad arrivare a cadere nuovamente. Il programma seguente simula tutto ciò:

PROGRAMMA "PALLINA CHE RIMBALZA"

```
LIST

10 PRINT CHR$(147);CHR$(158)
20 POKE 53280,2 : POKE 53281,0
30 S=54272 : POKE S+24,15
40 POKE S+5,0 : POKE S+6,240
50 POKE S,0 : POKE S+1,80
60 R=5 : C=22 : H=1 : V=1
70 POKE 214,R : PRINT : POKE 211,C
80 PRINT CHR$(113)
90 FOR T=1 TO 20 : NEXT T
100 POKE 214,R : PRINT : POKE 211,C
110 PRINT " "
120 V=V+0.2 : R=R+V : C=C+H
125 IF R>23 THEN 125
130 IF C>1 AND C<38 THEN 160
140 POKE S+4,33 : FOR T=1 TO 20
150 NEXT T : POKE S+4,32 : H=-H
160 IF R<20 THEN 70
170 POKE S+4,33 : V=-V : FOR T=1 TO 20
180 NEXT T : POKE S+4,32 : GOTO 70
READY.
```

In questo programma il fattore di gravità è scritto alla riga 120: sommare 0.2 a V significa che la variazione di R (la posizione verticale) non è più costante, ma aumenta ad ogni ciclo accelerando la pallina.
Quando la pallina raggiunge la parte superiore dello schermo viene emesso un suono, e la sua direzione è invertita dalla riga 170. V diventa allora negativa, diminuendo continuamente il numero di riga; l'aggiunta del fattore di gravità alla riga 120 rende però V sempre meno negativa, rallentando il movimento verso l'alto finché esso cessa; V ritorna positiva e la pallina comincia a muoversi ancora una volta verso il basso.
Questa schermata dimostra come si muove la pallina con questo programma (ancora una volta, questo è quello che vedreste se fermaste il computer e non cancellaste le immagini precedenti, neutralizzando la linea 110 con l'aggiunta di un REM):

VISUALIZZAZIONE DELLA PALLINA CHE RIMBALZA

La pallina rimbalza tutto attorno come prima ma essa non raggiunge più la stessa altezza ad ogni rimbalzo: quest'ultima decresce gradualmente, benché il movimento orizzontale rimanga inalterato. Il risultato di ciò è un rozzo esempio di una curva detta "parabola".
La pallina potrebbe eventualmente raggiungere la base dello schermo quando il programma finisce in un ciclo senza fine.
Nello stesso modo in cui potete influenzare il movimento verticale per mezzo della gravità, potete anche alterare il movimento orizzontale; questo fatto fornisce l'impressione di un oggetto il quale, non solo cade sotto l'azione della forza di gravità, ma che si trova anche in balia di un forte vento.

Simulare la gravità in alta risoluzione

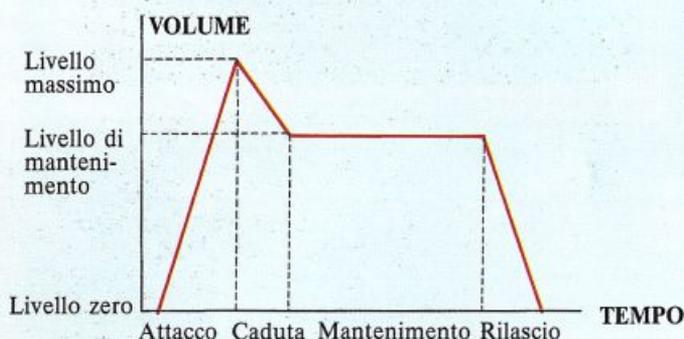
La curva descritta dalla pallina nel programma di simulazione della gravità non è molto regolare dal momento che la pallina stessa è visualizzata come un carattere di un testo, e ci sono solo 40x25 posizioni che essa può occupare.
Se volete ottenere dei rimbalzi più regolari, potete provare a riprodurre la traiettoria della pallina in alta risoluzione, disegnando un singolo punto per ogni determinata coppia di coordinate grafiche. Questo fatto permette movimenti più regolari che si sviluppano su una griglia 300x200.
Per fare tutto ciò, comunque, dovete modificare il programma in modo tale che le coordinate in bassa risoluzione siano ovunque sostituite dalle corrispondenti in alta risoluzione; se fate riferimento alla griglia di pag. 58 non dovreste trovare difficoltà in questa operazione.
Una volta trasportato in alta risoluzione, il programma lavorerà più lentamente della versione precedente, ma le curve ottenute risulteranno più realistiche.

COSTRUIRE I SUONI

Come avete visto nel primo volume, il Commodore possiede dei comandi particolarmente potenti per la generazione di suoni, che vi permettono di generare un vasto insieme di note e di effetti sonori: il Sound Interface Device (dispositivo di interfaccia sonora), detto chip SID, può fare ben più che non produrre semplicemente un suono ad una particolare frequenza. In queste pagine troverete altri particolari su come controllare il profilo di un suono, una caratteristica che è conosciuta come "l'inviluppo" sonoro, o ACMR.

Come un suono varia nel tempo

La forma dell'inviluppo di un suono è un grafico che mostra come varia la sua intensità mentre il suono si sviluppa dall'inizio alla fine. Sul Commodore quattro parametri possono essere modificati per modellare l'intensità sonora: Attacco, Caduta, Mantenimento e Rilascio.

LA FORMA DI UN SUONO TIPICO

L'Attacco è il tempo che il volume impiega a passare dal livello zero al suo livello massimo; può assumere un valore qualsiasi compreso tra 2 millisecondi e 8 secondi. La Caduta è quella parte dell'inviluppo che inizia alla fine del tempo di Attacco e termina quando il volume ha raggiunto il suo valore di mantenimento; può assumere un valore qualsiasi tra 6 millisecondi e 24 secondi.

Il Mantenimento è differente dagli altri tre parametri in quanto non è un tempo ma un livello di volume, espresso come percentuale del livello massimo raggiunto al termine della fase di Attacco; il valore 0 indica che il volume decadrà ad un livello di Mantenimento di 0, mentre un valore di 15 indica che il livello di Mantenimento sarà identico a quello massimo, cioè il volume non decadrà affatto.

Il Rilascio è il tempo necessario affinché il volume passi dal suo valore di Mantenimento al livello 0, e può variare da 6 millisecondi a 24 secondi. Per programmare tutti questi fattori, dovete selezionare i valori appropriati dei parametri ACMR.

Ecco un programma che dimostra gli effetti dei parametri ACMR. Esso suona ripetutamente la stessa nota, cambiando ogni volta i valori ACMR:

VALORI DEI PARAMETRI ACMR

La tabella mostra i valori relativi alle possibili predisposizioni da 0 a 15 dei parametri ACMR

Predisposizione	Tempo di Attacco (sec.)	Tempo di Caduta (sec.)	Livello di Mantenimento (%)	Tempo di Rilascio (sec.)
0	0.002	0.006	0	0.006
1	0.008	0.024	7	0.024
2	0.016	0.048	13	0.048
3	0.024	0.072	20	0.072
4	0.038	0.114	27	0.114
5	0.056	0.168	33	0.168
6	0.068	0.204	40	0.204
7	0.08	0.24	47	0.24
8	0.1	0.3	53	0.3
9	0.25	0.75	60	0.75
10	0.5	1.5	67	1.5
11	0.8	2.4	73	2.4
12	1	3	80	3
13	3	9	87	9
14	5	15	93	15
15	8	24	100	24

PROGRAMMA DI DIMOSTRAZIONE ACMR

```
LIST -180

10 PRINT CHR$(147)
20 S=54272
30 DIM N(11,2) : POKE S+3,8
40 RESTORE : FOR C=0 TO 11
50 READ N(C,0)
60 READ N(C,1)
70 READ N(C,2)
80 NEXT C : POKE S+24,15
90 FOR C=1 TO 4
100 READ W
110 READ AD
120 READ SR
130 POKE S+5,AD
140 POKE S+6,SR
150 FOR K=0 TO 11
160 POKE S,N(K,1)
170 POKE S+1,N(K,0)
180 POKE S+4,W+1

READY.
```

```
LIST 190-

190 FOR T=1 TO N(K,2) : NEXT T
200 POKE S+4,W
210 NEXT K
220 FOR T=1 TO 1000 : NEXT T
230 NEXT C
240 DATA 25,29,160,25,29,720
250 DATA 21,30,160,22,96,80
260 DATA 25,29,160,28,49,160
270 DATA 31,164,80,33,134,160
280 DATA 28,49,80,31,164,160
290 DATA 25,29,160,22,96,960
300 DATA 64,9,0
310 DATA 32,144,243
320 DATA 16,9,163
330 DATA 64,88,57
READY.
```

Programmare l'inviluppo di un suono
Ognuno dei tre canali sonori del Commodore è associato a due registri di controllo ACMR.
L'Attacco e la Caduta sono controllati da uno dei registri del chip SID, il Mantenimento e il Rilascio dall'altro.

REGISTRI ACMR

Ogni registro è costituito da due semi-byte o "nibble". Questi nibble controllano caratteristiche diverse del suono.

Registro SID (S=54272)	Canale Sonoro	Parametro ACMR Nibble alto	Nibble basso
S+5	1	Attacco	Caduta
S+6	1	Mantenimento	Rilascio
S+12	2	Attacco	Caduta
S+13	2	Mantenimento	Rilascio
S+19	3	Attacco	Caduta
S+20	3	Mantenimento	Rilascio

I registri del chip SID sono divisi in due parti uguali, ciascuna di quattro bit: Attacco, Caduta, Mantenimento e Rilascio sono controllati ciascuno da uno di questi nibble: avendo a disposizione 4 bit, i valori possibili sono 16. Caduta e Rilascio sono controllati dal nibble meno significativo, perciò assegnare a questi parametri il valore 10, significa avere un nibble di valore 10. Attacco e Mantenimento sono invece controllati dal nibble alto per cui, assegnare a questi parametri il valore 10, significa che il nibble relativo ha valore 160.

PREDISPOSIZIONI POSSIBILI DEI PARAMETRI ACMR

Caduta e Rilascio sono controllati dai nibbles meno significativi, così il Valore della Predisposizione e il valore del nibble stanno nello stesso intervallo (0–15). Attacco e Mantenimento sono controllati dai nibbles più significativi e così devono essere convertiti nelle Predisposizioni possibili da 0 a 15.

Predisposizione	Valore di Attacco/Mantenimento	Valore di Caduta/Rilascio
0	0	0
1	16	1
2	32	2
3	48	3
4	64	4
5	80	5
6	96	6
7	112	7
8	128	8
9	144	9
10	160	10
11	176	11
12	192	12
13	208	13
14	224	14
15	140	15

Supponete di voler programmare, usando il canale sonoro 2, un suono che possiede il parametro di Attacco impostato a 6 e quello di Caduta a 11. Dalla tabella precedente potete notare che un Attacco di 6 corrisponde al valore 96 e una Caduta di 11 corrisponde al valore 11.

COME UN BYTE CONTROLLA DUE PARAMETRI

Sommando i valori dei due nibble in uno specifico byte, potete ottenere un registro ACMR che controlla due diverse caratteristiche di un suono.

	Nibble di ATTACCO				Nibble di CADUTA			
Valori dei bit	128	64	32	16	8	4	2	1
	0	1	1	0	1	0	1	1

Valore del Nibble di ATTACCO = 96 Valore del Nibble di CADUTA = 11

Valore totale del byte ATTACCO/CADUTA = 107

Sommando questi valori si ottiene 107, il quale deve essere caricato nel registro SID S+12, che è il registro che controlla Attacco e Caduta sul canale 2. Ecco un programma che mostra la varietà di suoni che questi parametri possono produrre; esso vi permette di controllare la forma d'onda e i parametri ACMR del suono prodotto:

PROGRAMMA ACMR CONTROLLATO DA TASTIERA

```
LIST -180

10 S=54272
20 DIM N(11,2) : POKE S+3,8
30 RESTORE : FOR C=0 TO 11
40 READ N(C,0)
50 READ N(C,1)
60 READ N(C,2)
70 NEXT C : POKE S+24,15
80 PRINT CHR$(147)
90 PRINT "QUALE FORMA D'ONDA"
100 PRINT "1=TRIANGOLARE      3=IMPULSO"
110 PRINT "2=DENTE DI SEGA    4=RUMORE CA
SUALE"
120 INPUT "BATTI UN TASTO DA  1 A 4";W
130 INPUT "VALORE DI ATTACCO  (0-15)";A
140 INPUT "VALORE DI CADUTA   (0-15)";D
150 INPUT "LIVELLO DI MANTENIMENTO (0-15
)";L
160 INPUT "VALORE DI RILASCIO (0-15)";R
170 AD=16*A+D : SR=16*L+R
180 POKE S+5,AD : POKE S+6,SR

READY.
```

```
LIST 190-

190 FOR K=0 TO 11
200 POKE S,N(K,1)
210 POKE S+1,N(K,0)
220 POKE S+4,2↑(W+3)+1
230 FOR T=1 TO N(K,2) : NEXT T
240 POKE S+4,2↑(W+3)
250 NEXT K
260 GOTO 80
270 DATA 25,29,160,25,29,720
280 DATA 21,30,160,22,96,80
290 DATA 25,29,160,28,49,160
300 DATA 31,164,80,33,134,160
310 DATA 28,49,80,31,164,160
320 DATA 25,29,160,22,96,960
READY.
```

EFFETTI SONORI SOFISTICATI

Ora siete pronti per dare un'occhiata ai comandi del Commodore per ottenere suoni sofisticati, che riguardano filtri e modulazione ad anello. Può essere molto arduo impadronirsi di queste tecniche e, comunque, acquisterete una totale familiarità con esse solo dopo molte ore di sperimentazione; i particolari contenuti in queste due pagine vi condurranno nella giusta direzione, dopo di che potrete effettuare gli esperimenti che più vi interessano: basandovi sui comandi del chip SID, avete davanti a voi molte ore di interessanti scoperte.

Filtrare un effetto sonoro
Fatta eccezione per le note pure (chiamate suoni sinusoidali), tutti i suoni sono costituiti da molte frequenze. Una di queste, chiamata "fondamentale", domina sulle altre e costituisce la frequenza caratteristica del suono; le altre sono multipli di questa frequenza e vengono dette "armoniche": la seconda armonica ha una frequenza doppia della fondamentale, la terza armonica tripla, e così via.
Le caratteristiche di un suono possono essere drasticamente variate alterando l'ampiezza di un piccolo numero di sue armoniche e, all'interno del chip SID, ciò può essere effettuato per mezzo di filtri. All'uscita di ogni canale sarà innanzi tutto assegnata una frequenza, poi una forma d'onda, e quindi essa verrà modificata dai valori dei parametri ACMR, i quali sono immessi nella sezione mixer; le uscite di tutte e tre i canali sono infine portati all'ingresso di una sezione di filtro e controllo principale di volume, la quale fornisce il suono finale.

IL SUONO È MIXATO E FILTRATO

Come potete notare dal disegno, qualsiasi tipo di filtraggio effettuato, influenza simultaneamente le uscite di tutti e tre i canali. All'interno del chip SID sono disponibili tre diversi tipi di filtri, i quali possono essere usati separatamente o in una combinazione qualsiasi; essi sono conosciuti come: passa basso, passa banda e passa alto. Ognuno di questi tre filtri possiede una frequenza di "taglio", che è il punto dell'insieme di tutte le frequenze in cui esso comincia ad operare l'eliminazione effettiva di una parte del suono.

CARATTERISTICHE DEI FILTRI AUDIO

Filtro passa alto

Filtro passa basso

Filtro passa banda

I filtri sono controllati da alcuni bit memorizzati in quattro registri all'interno del chip SID e numerati da S+21 a S+24. Le locazioni S+21 e S+22 formano un unico registro ad 11 bit (tre in S+21 e otto in S+22), il quale controlla la frequenza di taglio nell'insieme dei valori che vanno da 30 Hz a 12KHz; S+23 controlla quale canale sonoro deve essere filtrato (i 3 bit meno significativi) e la rapidità della caratteristica di ogni filtro (i 4 bit più significativi); S+24 controlla il filtro attivo e il volume principale.
I programmi seguenti generano del rumore bianco su un canale sonoro e quindi vi applicano il filtro passa banda, cambiando continuamente la sua frequenza di taglio, in modo che voi possiate ascoltarne gli effetti sul suono prodotto:

PROGRAMMA "RAZZO"

PROGRAMMA "ARMA DA FUOCO"

```
LIST
10 PRINT CHR$(147)
20 S=54272
30 POKE S+5,0
40 POKE S+6,240
50 POKE S,0
60 POKE S+1,20
70 POKE S+4,33
80 POKE S+23,241
90 POKE S+24,47
100 FOR K=1 TO 3
110 FOR C=0 TO 160 STEP 20
120 POKE S+22,C
130 NEXT C
140 FOR C=160 TO 0 STEP -2
150 POKE S+22,C
160 NEXT C
170 NEXT K
180 POKE S+4,16
190 FOR T=1 TO 1000:NEXT
200 GOTO 10
READY.
```

Modificare i suoni con la modulazione ad anello

La modulazione ad anello è un processo per cui la forma d'onda triangolare all'uscita di un canale sonoro è sostituita da una combinazione di quest'ultima e dell'uscita del canale seguente; così, applicando la modulazione ad anello al canale 1, la sua uscita triangolare sarà sostituita da una combinazione, modulata ad anello, delle uscite dei canali 1 e 2. La modulazione ad anello, applicata al canale 2, coinvolgerà le uscite dei canali 2 e 3.

Potete selezionare questo effetto portando a "1" il terzo bit nel registro di controllo del canale e occorre perciò utilizzare la mascheratura dei bit: il terzo bit ha valore 4, perciò, per portarlo a "1", dovete ricaricare il valore precedente dopo aver effettuato l'OR con 4 in modo che venga modificato solo tale bit. La riga seguente effettua questa operazione:

```
POKE S+CR, PEEK(S+CR) OR 4
```

CR è il numero del registro di controllo del chip SID per il canale scelto. AND azzererà lo stesso bit:

```
POKE S+CR, PEEK(S+CR) AND 251
```

Ecco due programmi che vi permettono di ascoltare la modulazione ad anello al lavoro; il primo simula il suono di un campanello e il secondo di una sirena d'allarme:

REGISTRI PER LA MODULAZIONE AD ANELLO

Canale	Registro di controllo
1	S+4
2	S+11
3	S+18

PROGRAMMA "CAMPANELLO"

PROGRAMMA "SIRENA D'ALLARME"

Attenzione a questo banale errore

Mentre state lavorando con programmi che emettono suoni, potreste scoprire che alcuni di loro non sembrano proprio funzionare benché il testo del programma sia privo di errori. Il vostro problema potrebbe essere dovuto ai programmi eseguiti in precedenza che hanno usato dei registri del chip SID che ora interferiscono con il nuovo programma; se ciò accade, la cosa migliore da fare è spegnere il computer per azzerare il chip SID, e quindi immettere nuovamente il programma.

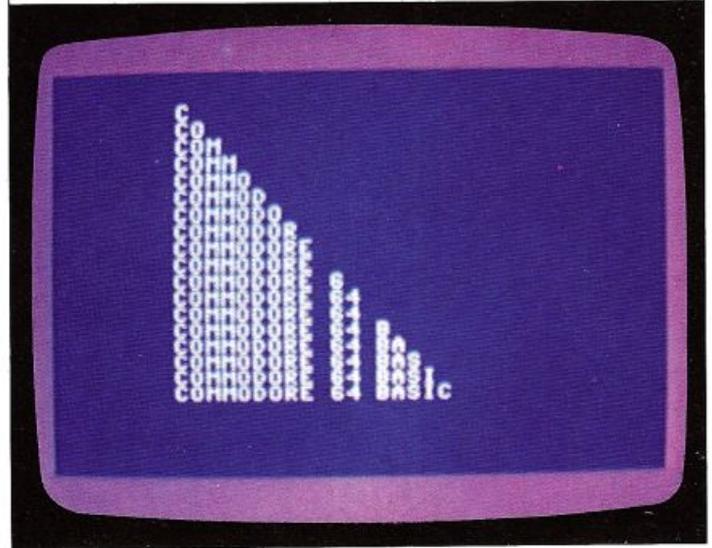
LAVORARE CON LE PAROLE

Fino ad ora avete trattato le stringhe di caratteri, o le parole che formano le stringhe, come unità indivisibili. Alcuni dei programmi precedenti hanno sommato delle stringhe, ma nessuno di essi ha guardato tra le virgolette poste all'inizio e alla fine di ogni stringa per lavorare sui caratteri che la compongono; con il Commodore potrete prendere delle stringhe separatamente e riassemblare in diversi modi i caratteri che le costituiscono. Ciò significa che potete programmare il computer in maniera tale che esso prenda parte di una parola, o un gruppo di parole, e le esamini: operazione, questa, che può rivelarsi di grande utilità. Come la maggior parte dei computer che usano il linguaggio BASIC, il Commodore possiede una famiglia di comandi che possono essere usati per manipolare le stringhe. Alcuni dei più comuni sono: LEFT$, RIGHT$, MID$; essi sono usati per estrarre rispettivamente i primi, gli ultimi e i caratteri di mezzo di una stringa.

Come troncare le parole
Voi potete far sì che il Commodore tronchi una parola, facendogli selezionare una parte della stringa. Questa operazione è abbastanza semplice; per scoprire come fare, innanzi tutto, immettete questo programma:

PROGRAMMA "SEZIONATORE DI STRINGHE"

Una tecnica particolare è usata alla riga 50, dove un comando LEFT$ appare come parte di una espressione riguardante una stringa: per ogni valore di N, la riga 50 visualizza una stringa lunga N caratteri, partendo dal primo della stringa originale, in questo caso "C" fino, all'Nesimo carattere. In questo modo la prima stringa è "C", la seconda "CO" e così via, finché N non diventa uguale alla lunghezza della stringa predefinita. Con questo programma potete usare qualsiasi stringa ma è bene che N non superi la lunghezza di una linea di schermo (40 caratteri); se usate una stringa diversa da quella mostrata, assicuratevi che il valore massimo di N (riga 40) sia uguale alla lunghezza della vostra stringa. Ecco la visualizzazione proposta dal programma con la stringa impostata precedentemente alla riga 20:

RISULTATO DEL "SEZIONATORE DI STRINGA"

Potete usare questo tipo di tecnica per selezionare stringhe che inizino tutte con lo stesso carattere o la stessa parola e, ad esempio, visualizzarle in una serie di liste.
L'effetto opposto è molto facile da realizzare; provate ad aggiungere queste righe al programma precedente:

```
70 FOR N=1 TO 18
80 PRINT TAB (8); RIGHT$(A$,19-N)
90 NEXT N
```

Ora, mentre N è incrementato da 1 a 18, la lunghezza della stringa emessa è decrementata da 18 a 1, e le lettere vengono eliminate dalla sinistra.

Come selezionare parti di una frase
Ora potete esplorare questa tecnica. Il prossimo programma mostra come il Commodore può selezionare parti di una stringa e usarle in modi diversi:

SELEZIONATORE SELETTIVO DI STRINGHE

Come potete vedere dalla figura che esso riproduce, non siete limitati ad agire sui primi N caratteri della stringa; infatti potete prendere qualsiasi gruppo di caratteri consecutivi contenuti in una parola o in una frase. In questo programma la riga 60 lavora allo stesso modo della riga 50 del primo programma. La riga 90 genera una stringa di 6 caratteri, prelevando i caratteri dall'ottavo al quattordicesimo di A$ ed infine, la riga 120 crea una terza stringa con gli ultimi 6 caratteri. Sebbene queste tre sottostringhe siano formate dai caratteri di A$, la stessa A$ rimane intatta. Questo vi consente di prendere un gruppo di parole e scegliere proprio quelle che vi servono in un programma.

Giocare con le parole usando i comandi per le stringhe
Il prossimo programma vi mostra come potete usare questi metodi di manipolazione di stringhe in un gioco. Si tratta della versione computerizzata del conosciutissimo "gioco dell'impiccato", in cui un giocatore scrive una parola o una frase e l'altro deve indovinarla; il computer visualizza le lettere indovinate nella corretta posizione e, inoltre, vi consente di cercare di indovinare l'intera parola:

PROGRAMMA "GIOCO DELL'IMPICCATO"

```
LIST -110

10 POKE 53280,0 : POKE 53281,0 : PRINT C
HR$(147) : PRINT TAB(14);"L'IMPICCATO"
20 POKE 214,8 : PRINT : PRINT "CHIEDI A
UN AMICO DI SCRIVERE UNA PAROLA"
30 PRINT TAB(7);"O UNA FRASE DA INDOVINA
RE"
40 PRINT : PRINT TAB(5);"** NON GUARDARE
 LO SCHERMO! **"
50 POKE 214,17 : PRINT : PRINT "   PREMI
 UN TASTO QUANDO HAI FINITO"
60 PRINT : PRINT TAB(13); : INPUT "PAROL
A = ";A$
70 PRINT CHR$(147): PRINT "L'IMPICCATO -
 PREMI  1  PER INDOVINARE"
80 P=(32-LEN(A$))/2 : S=0 : POKE 214,7 :
 PRINT
90 FOR N=1 TO LEN(A$) : POKE 211,P+N
100 IF MID$(A$,N,1)=" " THEN PRINT " ";
110 PRINT "-"; : NEXT N

READY.
```

```
LIST 120-

120 POKE 214,18 : PRINT : PRINT TAB(9);
: INPUT "PROVA UNA LETTERA";T$
130 IF T$="1" THEN 180
140 S=S+1 : POKE 214,13 : PRINT : PRINT
TAB(12);"TENTATIVI =";S : POKE 214,7
150 PRINT : FOR N=1 TO LEN(A$)
160 IF T$=MID$(A$,N,1) THEN POKE 211,P+N
 : PRINT T$;
170 NEXT N : GOTO 120
180 POKE 214,22 : PRINT : PRINT TAB(7);
: INPUT "PROVA A INDOVINARE";T$
190 PRINT CHR$(147)
200 POKE 214,10 : PRINT : POKE 211,14
210 IF T$=A$ THEN PRINT "**ESATTO**"
220 IF T$<>A$ THEN PRINT "--SBAGLIATO--"

230 FOR T=1 TO 5000 : NEXT T : GOTO 10
READY.
```

Le righe da 10 a 60 visualizzano i titoli iniziali. Quando un amico ha immesso la stringa che dovete indovinare, la riga 80 calcola la lunghezza della stessa, usando il comando LEN, e imposta il punteggio (S) a zero.
Il programma ora deve visualizzare dei simboli che rappresentano le lettere della stringa di prova.
Quando indovinate una lettera, questa andrà a sostituirsi ai simboli, inoltre, per permettere di usare frasi, e non singole parole, nella stringa di prova saranno evidenziati gli spazi tra le parole.
La riga 110 visualizza dei trattini che rappresentano i caratteri mentre la riga 90 usa il valore di P per fare in modo che la visualizzazione dei trattini sia centrata nella linea di schermo (un effetto simile lo si può trovare in un programma di word-processing).
Se volete cercare di indovinare l'intera parola o frase, invece di immettere una sola lettera, premete 1. Il programma salta alla riga 180 e la parola, o la frase, che immettete (T$) è comparata con la stringa memorizzata: quindi viene visualizzato il messaggio "esatto" oppure, se il tentativo non ha avuto successo, il messaggio "sbagliato".
Quando viene tentata una sola lettera, le righe da 150 a 170, la comparano con ognuno dei caratteri della stringa memorizzata; se il tentativo ha successo, la lettera è visualizzata nella giusta posizione.

GIOCO DELL'IMPICCATO

Potete facilmente limitare il numero di tentativi, aggiungendo l'istruzione:

```
IF S>N THEN STOP
```

dopo la riga in cui è calcolato il valore di S. Se commettete un errore di immissione nel programma, può risultare difficoltoso interromperlo usando i tasti STOP e RESTORE; per facilitare l'interruzione del programma aggiungete una ulteriore riga di controllo del valore di T$:

```
145 IF T$="2" THEN STOP
```

Il programma si arresterà premendo il tasto 2.

SCRIVERE I GIOCHI 1

Le sei pagine seguenti vi guideranno nello scrivere un video gioco, mostrandovi come assemblare le varie parti per costruire un programma completo. Scrivere un gioco richiede una attenta pianificazione prima di cominciare veramente a scrivere righe di programma; tanto per cominciare è necessario decidere che tipo di gioco desiderate: molti giochi combinano la vostra abilità con degli elementi casuali (bisogna tirare i dadi, girare una carta ecc.), mentre molti altri sono composti da un certo numero di fasi di gioco.
Per programmare un videogioco, è meglio cominciare a tracciare uno schizzo approssimativo delle varie schermate, scegliendo i colori e le posizioni di ogni simbolo o carattere fisso; dovrete riferirvi spesso a questo schizzo durante la stesura del programma. Potreste inoltre stendere un diagramma di flusso (flow chart) che riporti le varie fasi del programma e l'ordine con cui esse verranno eseguite; non è necessario preparare un flow chart dettagliato, è sufficiente una serie di operazioni collegate da frecce che indicano la loro esatta sequenza. Un programma per un videogioco completo risulterà molto più complicato di qualunque altro che avete scritto fino ad ora, perciò è bene spendere del tempo per pianificarlo prima di immetterlo effettivamente.

Immettere la prima parte del gioco
Per il video gioco riportato su queste pagine, la parte di pianificazione è già stata completata e voi potete immettere subito la prima parte di ciò che diventerà un programma in due fasi. Il programma seguente è un gioco molto semplice: uno di quelli a cui chiunque è in grado di giocare senza che sia necessario possedere particolari conoscenze sul computer o sul gioco stesso. La prima parte del gioco implica il dover centrare un'astronave mobile; siccome il programma contiene sia caratteri che sprite personalizzati, ricordatevi di spostare l'area di memoria BASIC usando le tecniche di pag. 14, in modo da far posto per questi ultimi prima di immettere il programma:

PARTE 1: SCHERMATA 1

```
10 PRINT CHR$(147) : POKE 53280,6
20 POKE 53281,12 : V=53248 : POKE V+21,3
30 POKE 2040,48 : POKE 2041,49
40 POKE V+39,7 : POKE V+40,2
50 FOR N=0 TO 127 : READ BYTE
60 POKE 3072+N,BYTE : NEXT N
70 POKE 56334,PEEK(56334) AND 254
80 POKE 1,PEEK(1) AND 251
90 FOR N=0 TO 511 : POKE 2048+N,PEEK(532
48+N) : NEXT N
100 POKE 1,PEEK(1) OR 4 : POKE 56334,PEE
K(56334) OR 1
110 POKE 53272,(PEEK(53272) AND 240)+2
120 FOR N=32 TO 39 : READ P
130 POKE 2568+N,P : NEXT N
140 S=54272 : POKE S+24,15
150 POKE S+5,0 : POKE S+6,240
160 POKE S,0 : POKE S+1,80
170 TI$="000000" : N=0 : F=0 : Q=0
180 R=INT(RND(1)*15) : C=INT(RND(1)*26)
190 L=17 : M=16 : PRINT CHR$(147)

READY.
```

Il programma fornisce una postazione laser che potete muovere a destra e a sinistra mediante i tasti X e Z; potete anche sparare, ma solo in linea retta sullo schermo: un certo numero di astronavi si avvicineranno e voi dovete distruggerle tutte per poter proseguire il gioco. Il programma partirà dopo aver battuto RUN e premuto un tasto qualsiasi; siccome esso contiene molte istruzioni di DATA e di POKE, impiegherà un certo tempo a inizializzare lo schermo: non pensate di aver commesso un errore se non succederà nulla per pochi secondi.

PARTE 1: VARIABILI

La prima parte del gioco utilizza 16 variabili per controllare i disegni e memorizzare i colpi andati a segno.

Variabile	Utilizzo
A	Imposta la direzione dell'astronave
CS	Memorizza i caratteri battuti dal giocatore
F	Memorizza il totale dei colpi sparati dal laser
H	Indica i colpi andati a segno
L,M	Coordinate della postazione laser
N,P	Variabili di uso generale
Q	Memorizza il numero di volte che il laser ha sparato
R,C	Coordinate dell'astronave
S	Indirizzo base del chip SID
SC	Memorizza il punteggio relativo a questa fase
T	Imposta il ciclo di ritardo
X,Y	Controlla le coordinate del laser (1=destra, -1=sinistra)

La seconda schermata relativa al programma contiene i numeri di riga che fanno prendere le giuste decisioni al computer e lo indirizzano alle subroutine successive. Noterete che, procedendo nel programma, i numeri di riga talvolta differiscono di più di 10; in questo modo è più semplice identificare le subroutine:

PARTE 1: SCHERMATA 2

```
200 N=N+1 : IF N=6 THEN 400
210 H=0 : A=4 : C=C+10
220 FOR T=1 TO RND(1)*900 : NEXT T
230 X0=INT((M+2)/32) : X1=INT((C+2)/32)
240 POKE V+16,X0+2*X1
250 POKE V,(M+2)*8-256*X0 : POKE V+1,(L+
7)*8
260 POKE V+2,(C+2)*8-256*X1 : POKE V+3,(
R+7)*8
270 POKE S+4,33 : FOR T=1 TO 20
280 NEXT T : POKE S+4,32
290 GET C$ : IF C$="Z" THEN M=M-1
300 IF C$="X" THEN M=M+1
310 IF M<2 THEN M=2
320 IF M>37 THEN M=37
330 IF C$="M" THEN GOSUB 500
340 IF H=1 THEN 180
350 X=C : C=C+A
360 IF C>35 OR C<4 THEN A=-A
370 P=INT(RND(0)*10+1)
380 IF P=3 THEN GOSUB 600
390 GOTO 230

READY.
```

Le righe 290 e 300 spostano la postazione del laser in entrambe le direzioni possibili (tasti X e Z) e le righe 310 e 320 gli impediscono di uscire dallo schermo; la postazione si muove di una posizione ogni volta che premete un tasto ma, se immettete il comando diretto POKE 650,128 prima di eseguire il programma, il movimento verrà ripetuto automaticamente mantenendo premuto il tasto.
Se premete il tasto "M", il programma salterà alla routine di "fuoco" alla riga 500; essa visualizza una serie di caratteri predefiniti e, se viene colpito il bersaglio, il programma prepara un nuovo attacco.
Le righe 350 e 360 controllano il movimento dell'astronave; le righe 370 e 380 fanno saltare il programma alla routine di attacco nemico alla riga 600, una volta ogni 10 movimenti dell'astronave; questa operazione è effettuata estraendo un numero casuale tra 1 e 10: solo uno di questi valori, il 3, scatenerà la routine di attacco. La riga 390 continua lo stesso attacco risaltando alla riga 230.

La sezione delle subroutine

Infine ecco l'ultima parte del programma: essa contiene un paio di subroutine e i dati relativi ai disegni. Le righe da 510 a 560 scrivono e cancellano il raggio laser utilizzando i caratteri definiti nella prima schermata inoltre, se il valore di M è compreso tra C−1 e C+1, il raggio ha colpito il bersaglio. Le righe da 600 a 700 controllano il colpo di risposta dell'astronave: se esso cade sulla postazione laser, si udrà un'esplosione e verrà modificato il valore Q del vostro punteggio.
Le righe 400 e 410 calcolano il punteggio e terminano questa parte del programma; il punteggio è basato sul tempo impiegato a completare la prima fase, sul numero totale di colpi sparati e su quello dei colpi andati a segno; esso, in realtà, non appare in questa fase, lo incontrerete più avanti nello sviluppo del programma.
Una volta che avete immesso tutte le righe di programma delle schermate seguenti, salvatele (SAVE) su nastro o disco prima di eseguirle, in modo che, se avete commesso errori tali da bloccare il computer, non perderete questa parte di programma e sarete pronti per aggiungervi la parte seguente.

PARTE 1: SCHERMATA 3

```
400 SC=TI/300+Q↑2-10*F
410 STOP
500 Q=Q+1 : FOR Y=16 TO R STEP -1
510 POKE 214,Y : PRINT : POKE 211,M
520 PRINT CHR$(31);CHR$(101)
530 POKE S+4,33 : POKE S+4,32
540 NEXT Y : FOR Y=16 TO R STEP -1
550 POKE 214,Y : PRINT : POKE 211,M
560 PRINT CHR$(31);" " : NEXT Y
570 IF M>C+1 OR M<C-1 THEN RETURN
580 POKE S+4,129 : FOR T=1 TO 200
590 NEXT T : H=1 : F=F+1 : POKE S+4,128
: RETURN
600 FOR Y=R+1 TO 17
610 POKE 214,Y : PRINT : POKE 211,X
620 PRINT CHR$(28);CHR$(101)
630 POKE S+4,33 : POKE S+4,32
640 NEXT Y : FOR Y=R+1 TO 17
650 POKE 214,Y : PRINT : POKE 211,X
660 PRINT CHR$(28);" " : NEXT Y
670 IF X>M+1 OR X<M-1 THEN RETURN
680 POKE S+4,129 : FOR T=1 TO 200

READY.
```

PARTE 1: SCHERMATA 4

```
LIST 690-

690 NEXT T : Q=Q+10 : POKE S+4,128
700 RETURN
750 DATA 0,0,0,0,0,0,0,24
760 DATA 0,0,24,0,0,24,0,0
770 DATA 24,0,0,24,0,0,126,0
780 DATA 7,255,224,31,255,248,31,255
790 DATA 248,3,255,192,0,255,0,127
800 DATA 255,254,192,0,3,191,255,253
810 DATA 117,85,86,95,255,250,118,219
820 DATA 110,54,219,108,31,255,248,0
830 DATA 0,24,0,0,255,0,0,165,128
840 DATA 3,255,192,60,0,60,255,255,255
850 DATA 169,36,149,169,36,149,255,255
860 DATA 255,48,0,12,15,255,240,1,126
870 DATA 128,3,24,192,6,0,96,4,0,32
880 DATA 12,0,48,30,0,120,0,0,0
890 DATA 0,0,0,0,0,0,0,0,0,0
900 DATA 24,24,24,24,24,24,24,24
READY.
```

Ecco il programma in azione. Nella prima immagine il laser sta sparando ad una astronave, mentre nella seconda, potete vedere reagire l'astronave:

PARTE 1: VISUALIZZAZIONI

SCRIVERE I GIOCHI 2

Nella seconda fase del gioco, la scena si sposta dallo spazio alla superficie marina dove una nave cerca di far esplodere un sottomarino mobile; anche in questo caso il vostro scopo è quello di colpire il nemico raggiungendo il migliore punteggio possibile. Le istruzioni relative al vostro punteggio non vengono ancora utilizzate in questa parte del programma, ma sono pronte per diventare operative non appena immetterete l'ultima parte del gioco, collegandola alle prime due.
Dovrete immettere tutte e tre le schermate relative a questa fase alla fine del programma che avete salvato per la fase 1; ricordatevi ancora una volta di salvare i programmi prima di eseguirli: siccome ci sono molti byte da caricare in memoria, un qualsiasi errore di battitura potrebbe far sì che vengano memorizzati in una locazione che potrebbe rivelarsi, ahimè fatale!
Anche questa parte di programma usa un certo numero di variabili per controllare movimenti e subroutine.

PARTE 2: VARIABILI

La seconda fase utilizza un totale di 6 variabili per controllare movimenti e subroutine.

Variabile	Utilizzo
B,D	Coordinate del sottomarino (B=riga, D=colonna)
C	Colonna a cui è posizionata la nave
F	Indicazione di carica di profondità sganciata (1=sganciata, 0=non sganciata)
U,E	Coordinate della carica di profondità (U=riga, E=colonna)

Predisporre lo scenario

La prima parte del programma produce uno sfondo colorato e imposta gli sprite relativi alla nave e al sottomarino:

PARTE 2: SCHERMATA 1

```
165 S=54272 : GOTO 1000
1000 FOR N=0 TO 127 : READ BYTE
1010 POKE 3072+N,BYTE : NEXT N
1020 V=53248 : PRINT CHR$(147);
1030 POKE 2040,48 : POKE V+39,7
1040 POKE 2041,49 : POKE V+40,0 : POKE V
+21,3
1050 FOR N=40 TO 55 : READ BYTE
1060 POKE 2568+N,BYTE : NEXT N
1065 POKE V+29,3
1070 N=0 : F=0 : TI$="000000"
1075 PRINT CHR$(154); : FOR P=1 TO 200
1080 PRINT "█"; : NEXT P : PRINT
1090 H=0*PEEK(V+31) : POKE V+39,7 : IF N
=5 THEN 1500
1100 POKE 53280,4 : POKE 53281,6
1110 C=145 : B=100+INT(RND(0)*100)
1120 D=INT(RND(0)*90)+50 : X0=0 : X1=0
1130 GET C$ : IF C$="Z" THEN C=C-8
1140 IF C$="X" THEN C=C+8
1150 IF C>305 THEN C=305
1160 IF C<30 THEN C=30

READY.
```

Non preoccupatevi, della riga 165: scoprirete il perché della sua presenza nella pagina seguente. Le righe 1075 e 1080 visualizzano un cielo azzurro sopra al mare, più scuro; la riga 1110 imposta la posizione orizzontale della nave e le righe da 1130 a 1190 controllano i suoi movimenti sull'acqua. Lo scopo di questo gioco è colpire il sottomarino: il tasto M comanda lo sganciamento della bomba di profondità della nave, che è manovrabile ancora per mezzo dei tasti X e Z; tutte queste funzioni sono controllate da una istruzione GET alla riga 1130, per ottenere una risposta rapida. La nave parte sempre dal centro dello schermo; la posizione del nemico è invece imprevedibile. Le righe 1110 e 1120 determinano una posizione di inizio casuale per il sottomarino e esso può comparire in una posizione qualsiasi.

Il programma principale e le sue subroutine

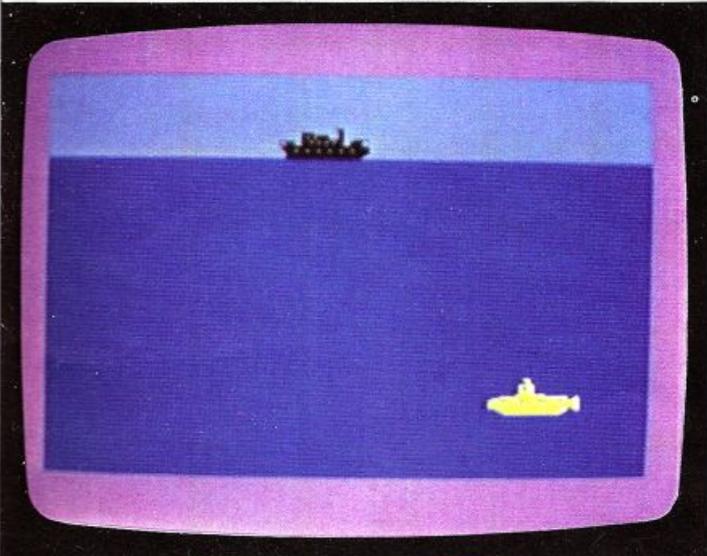
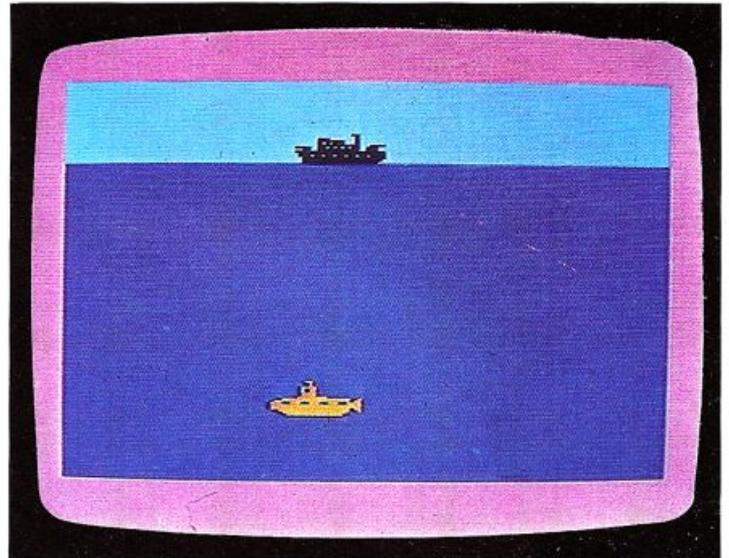
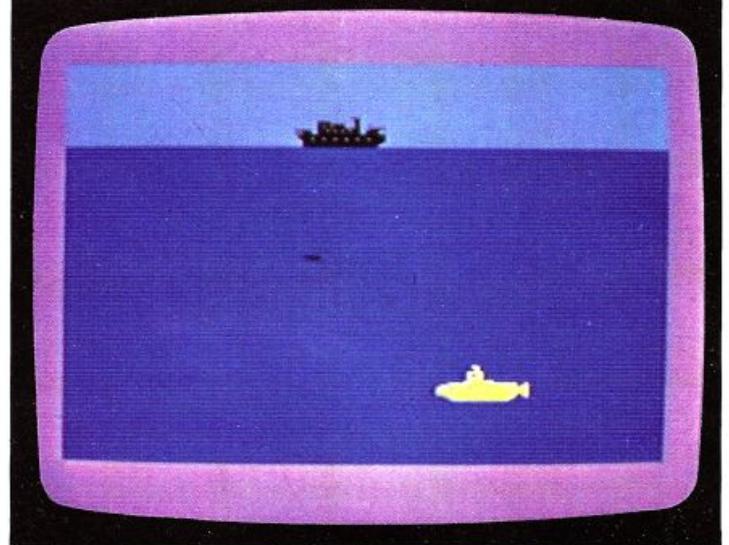
La seconda parte del programma contiene la maggior parte del programma principale, insieme con un certo numero di subroutine:

PARTE 2: SCHERMATE 2 E 3

```
1170 X1=INT(C/256) : POKE V+3,69
1180 POKE V+2,C-X1*256
1190 POKE V+16,X0+2*X1
1200 IF F=1 THEN GOSUB 1310
1205 IF H=1 THEN POKE V+39,8 : FOR T=1 T
O 1000 : NEXT T : GOTO 1090
1210 IF C$<>"M" OR F<>0 THEN 1230
1220 F=1 : SC=SC+100 : GOSUB 1300
1230 D=D-4 : X0=INT(D/256)
1240 POKE V,D-X0*256 : POKE V+1,B
1250 POKE V+16,X0+2*X1
1260 IF D<10 THEN D=350
1270 GOTO 1130
1300 U=4 : E=(C-10)/8
1310 POKE 214,U : PRINT : POKE 211,E
1320 PRINT " " : U=U+0.9
1330 IF U>22 THEN F=0 : RETURN
1350 POKE 214,U : PRINT : POKE 211,E
1360 PRINT CHR$(144);CHR$(102)
1370 IF (PEEK(V+31)AND1)=0 THEN RETURN
1380 N=N+1 : F=0 : H=1
1390 POKE S+4,129 : FOR T=1 TO 400

READY.
```

```
1400 NEXT T : POKE S+4,128
1410 POKE 214,U : PRINT : POKE 211,E
1420 PRINT " " : RETURN
1500 SC=SC+TI
1600 DATA 0,0,0,0,0,0,0,0,0,0,96,0
1610 DATA 0,32,0,0,32,0,0,240,0
1620 DATA 0,176,0,0,176,0,0,240,0
1630 DATA 0,240,0,1,254,0,31,255,249
1640 DATA 127,255,251,231,57,207
1650 DATA 255,255,255,255,255,255
1660 DATA 255,255,251,63,255,249
1670 DATA 31,255,240,15,255,224,0
1700 DATA 0,0,0,0,0,0,0,0,0,0,0,0
1710 DATA 0,0,0,0,0,0,0,0,1,128
1720 DATA 0,0,128,3,128,128,3,248,128
1730 DATA 2,184,128,3,255,143
1740 DATA 243,85,156,51,255,156
1750 DATA 255,255,255,127,255,255
1760 DATA 118,219,111,63,255,255
1770 DATA 63,255,254,63,255,252,0
1800 DATA 7,7,2,7,7,7,2,0,28,34
1810 DATA 193,0,7,136,112,0
READY.
```

La riga 1210 fa saltare il programma alla routine relativa alla bomba di profondità (riga 1300) se avete premuto "M". Il punteggio viene aggiornato ogni volta che viene sganciata una carica: esso dipende dal tempo trascorso grazie al jiffy clock (riga 1500); le righe da 1230 a 1260 controllano i movimenti e l'immagine del sottomarino; la riga 1270, infine, prosegue il programma facendolo ritornare alla riga 1130 per controllare se è stato premuto un tasto.
Le righe da 1300 a 1420 fanno muovere la carica di profondità sullo schermo; se essa raggiunge il fondo, le righe da 1380 a 1420 azzerano la variabile F e ritornano al programma principale.
Se la posizione della bomba coincide invece con quella del sottomarino (la rilevazione dello scontro tra sprite è alla riga 1370) l'attacco è terminato e ne comincia uno nuovo.
Quando eseguirete il programma, noterete che può essere sganciata una sola bomba di profondità alla volta: se la riga 1220 ha assegnato a F il valore 1, perché è stata sganciata una carica, la riga 1210 impedisce di sganciarne un'altra fino a che F non viene azzerata nuovamente; tutto ciò è valido sia che la carica raggiunga il fondo (riga 1330), sia nel caso che colpisca il sottomarino (riga 1380).
Le righe che vanno dalla 1600 alla 1770 contengono i dati relativi agli sprite mentre nelle due righe finali sono riportati i dati per la costruzione dei caratteri personalizzati: il primo è la carica di profondità mentre il secondo non è usato ma è disponibile per eventuali esperimenti.
Se sostituite il carattere rettangolare in reverse alla riga 1080 con CHR$ (103) vedrete uno sfondo di onde anziché un cielo terso; queste onde sono formate da un singolo carattere ricurvo per dare l'impressione di una superficie d'acqua. Tutti i byte contenuti nelle righe da 1600 a 1810 sono caricati in memoria dal ciclo alle righe da 1000 a 1060.
Ecco una sequenza di immagini tratte dal gioco; nell'ultima schermata il sottomarino è stato colpito ed ha cambiato colore:

PARTE 2: VISUALIZZAZIONI

PARTE 2: VISUALIZZAZIONI

La routine che aggiorna il punteggio
Il tempo è ancora una volta utilizzato per calcolare il punteggio alla fine del gioco. Se volete verificare il corretto funzionamento del programma, potete visualizzare il vostro punteggio per la fase 2 mediante la seguente istruzione:

```
1510 PRINT "PUNTEGGIO DELLA FASE 2="; SC
```

La riga 165 serve a saltare di netto la prima parte del programma; nella versione finale, dovrete eliminare questa riga ed aggiungere altre istruzioni per combinare i punteggi ottenuti nelle due diverse fasi del gioco.
Se avete raggiunto una certa conoscenza del funzionamento di queste due parti potrebbero interessarvi i punteggi fin qui ottenuti: provate allora a pensare ad un modo per combinare tali punteggi e per far sì che ciascuno di essi contribuisca per metà al punteggio finale.
Quando sarete sicuri che la seconda parte del programma funziona, sarete anche pronti per collegare le due parti assieme e per aggiungervi le istruzioni per giocare, facendone così un unico programma.

SCRIVERE I GIOCHI 3

Ora che avete immesso e salvato le prime due parti del gioco, siete pronti per aggiungervi le istruzioni e completare la parte che calcola il vostro punteggio. La riga 165, che era stata aggiunta nella seconda parte, non dovrà essere presente nella parte finale, perciò cancellatela prima di continuare.

Aggiungere le istruzioni per giocare

Se eseguite le prime due parti del programma, scoprirete che, benché esse costituiscano teoricamente un solo programma, si comportano ancora come due unità separate; quando state scrivendo il programma per un video gioco in più passi, come questo, ricordatevi che sarà necessaria qualche rifinitura alla versione finale perché essa funzioni nel modo migliore.

Collegare queste due parti è molto facile; modificate la riga 410 nel seguente modo:

```
410 POKE V+21,0 : GOTO 2200
```

Non si tratta di un errore, sebbene la parte successiva di programma cominci alla riga 1000; questo trucco vi permette anzi di aggiungere le istruzioni per il gioco a partire dalla riga 2200.

La nuova riga 410, inoltre, disattiva anche gli sprite usati nella prima parte.

Ora potete ritornare direttamente all'inizio e cominciare il programma con una schermata contenente il titolo, tutte le istruzioni di cui il giocatore ha bisogno e l'indicazione dei tasti che controllano gli oggetti che si muovono sullo schermo; dovrete spiegare al giocatore anche come iniziare il gioco in quanto, per tutto il tempo in cui compare questo messaggio, il programma è già in esecuzione.

La schermata seguente mostra le istruzioni prima della prima fase. La riga 5 fa saltare il programma oltre i due giochi, alle istruzioni mentre la riga 2140 permette al programma di ritornare all'inizio:

PARTE 1: ISTRUZIONI

```
5 GOTO 2000
2000 PRINT CHR$(147): POKE 53280,0 : POK
E 53281,0 : PRINT CHR$(5)
2010 POKE 214,2 : PRINT : POKE 211,13
2020 PRINT CHR$(5);"TIRO A SEGNO"
2030 PRINT TAB(13);"************"
2040 POKE 214,8 : PRINT : POKE 211,6
2050 PRINT "CINQUE ASTRONAVI ALIENE SONO
"
2060 PRINT TAB(3);"STATE AVVISTATE NEL V
OSTRO SETTORE" :PRINT
2070 PRINT TAB(6);"DOVETE DISTRUGGERE IL
 NEMICO":PRINT
2080 POKE 214,17 : PRINT : POKE 211,5
2090 PRINT "COMANDI DELLA POSTAZIONE LAS
ER"
2100 PRINT TAB(5);"Z=SINISTRA   X=DESTRA
   M=FUOCO" : PRINT : PRINT
2110 PRINT TAB(6);"PREMI UN TASTO PER CO
MINCIARE"
2120 GET C$ : IF C$="" THEN 2120
2130 GOTO 10
READY.
█
```

Le righe da 2000 a 2120 visualizzano il titolo del gioco e spiegano i suoi comandi: invece di cancellare lo schermo dopo un certo periodo di tempo, esso attende per un tempo determinato dall'istruzione GET: la riga 2130 fa in modo che, nel frattempo, il computer sia bloccato, continuando ad eseguire un ciclo sulla stessa istruzione. Questo ciclo viene ripetuto fino a che voi non immettete una stringa non nulla, in altre parole, fino a che non battete un carattere; in questo caso l'istruzione non viene più ripetuta e il programma salta alla riga 10, la quale cancella lo schermo ed inizializza il gioco:

PARTE 1: SCHERMATA DELLE ISTRUZIONI

```
             TIRO A SEGNO
             ************


    CINQUE ASTRONAVI ALIENE SONO
   STATE AVVISTATE NEL VOSTRO SETTORE

      DOVETE DISTRUGGERE IL NEMICO



     COMANDI DELLA POSTAZIONE LASER
     Z=SINISTRA   X=DESTRA   M=FUOCO

      PREMI UN TASTO PER COMINCIARE
```

Potete ora immettere le istruzioni relative alla seconda fase del gioco: queste ultime sono identiche alle precedenti e sono visualizzate al termine della prima fase alla riga 410. Ancora una volta viene usata l'istruzione GET per permettervi di incominciare a giocare solo quando sarete pronti; non appena premete un tasto, il programma salta alla riga 1000, che memorizza i dati relativi agli sprite, infine viene cancellato lo schermo:

PARTE 2: ISTRUZIONI

```
410 POKE V+21,0 : GOTO 2200
2200 PRINT CHR$(147) : POKE 53280,0 : PO
KE 53281,0 : PRINT CHR$(5)
2210 POKE 214,2 : PRINT : POKE 211,9
2220 PRINT CHR$(5);"CACCIA AL SOTTOMARIN
O"
2230 PRINT TAB(9);"*********************
"
2240 POKE 214,8 : PRINT : POKE 211,4
2250 PRINT "CINQUE SOTTOMARINI NEMICI HA
NNO"
2260 PRINT TAB(2);"INVASO LE VOSTRE ACQU
E TERRITORIALI" : PRINT
2270 PRINT TAB(4);"POTETE USARE BOMBE DI
 PROFONDITA" : PRINT
2280 POKE 214,17 : PRINT : POKE 211,11
2290 PRINT "COMANDI DELLA NAVE"
2300 PRINT : PRINT TAB(6); "Z=SINISTRA
X=DESTRA  M=FUOCO" : PRINT : PRINT
2310 PRINT TAB(6);"PREMI UN TASTO PER CO
MINCIARE"
2320 GET C$ : IF C$="" THEN 2320
2330 GOTO 1000
READY.
█
```

PARTE 2: SCHERMATE DELLE ISTRUZIONI

Potete naturalmente utilizzare i tasti che preferite per determinare i movimenti: è sufficiente sostituirli in tutto il programma. Ora nessuna delle due parti del gioco partirà prima che voi siate pronti e premiate un tasto per incominciare.

Come completare la subroutine per il punteggio

In primo luogo è necessario aggiungere una routine per ottenere il punteggio all'inizio del programma, mentre il punteggio della seconda fase è memorizzato da:

```
1500 LET SC=SC+TI
```

In ogni caso, se avete giocato separatamente alle due fasi del gioco e immettete l'istruzione PRINT SC dopo ogni fase, avrete certamente notato che il punteggio della prima delle due varia da -20 circa ad alcune centinaia o migliaia di punti mentre i risultati delle due prove dovrebbero essere più o meno dello stesso ordine di grandezza; potete fare ciò moltiplicando il punteggio finale della prima fase (riga 400) per 100:

```
400 LET SC=100*(TI/300+Q ↑ 2-10*F)
```

Questa riga è un buon test per verificare la vostra comprensione del significato delle variabili elencate da pag. 46 a pag. 49! Per rendere più interessante la visualizzazione del punteggio aggiungete poche righe per preparare una graduatoria; questo accorgimento è molto utile nei programmi di giochi in quanto un nuovo giocatore può farsi una idea di come il programma ha valutato la sua abilità: è molto meglio di un arido punteggio numerico che non vi dice nulla riguardo alla posizione occupata dal vostro risultato nella scala di tutti quelli possibili.
La visualizzazione di una graduatoria è adottata in una grande varietà di video giochi e, benché aggiungere questo tocco in più sia estremamente semplice, è proprio tale piccola aggiunta che segna la differenza tra un programma mediocre ed uno di cui potete essere veramente orgogliosi. Ecco la parte relativa al punteggio e una delle schermate che appariranno al termine del gioco:

SUBROUTINE PER IL PUNTEGGIO

```
LIST

1510 POKE V+21,0 : GOTO 2400
2400 SC=SC/5 : PRINT CHR$(147)
2410 POKE 53280,0 : POKE 53281,0 : PRINT
 CHR$(5)
2420 POKE 214,6 : PRINT
2430 PRINT TAB(6);"AVETE RAGGIUNTO IL GR
ADO DI"
2440 IF SC<1000 THEN A$="COMMODORO"
2450 IF SC>=1000 THEN A$="CAPITANO"
2460 IF SC>=2000 THEN A$="PILOTA"
2470 IF SC>=4000 THEN A$="CADETTO"
2480 IF SC>=6000 THEN A$="RECLUTA"
2490 POKE 214,8 : PRINT : POKE 211,15
2500 PRINT A$
2510 GOTO 2510
READY.
```

VISUALIZZAZIONE DEL PUNTEGGIO

Le righe da 2240 a 2480 suddividono il punteggio in fasce, a ognuna delle quali è assegnato un grado; una serie di istruzioni IF...THEN decide con quale grado il vostro punteggio entra nella graduatoria. Potete modificare i punteggi di suddivisione tra i gradi in modo da rendere il gioco più facile o più difficile (se vi sentite molto abili potete renderlo veramente difficile).
Avete ora completato un gioco in due fasi fornito di istruzioni, azione e una routine per il punteggio; benché le due fasi sviluppate in queste pagine siano relativamente semplici, il modo con cui sono state combinate può essere usato nei vostri video giochi personali anche più complessi: potete utilizzare questa tecnica a più stadi per assemblare un certo numero di sottoprogrammi scritti e provati separatamente. L'unica restrizione consiste nella dimensione della memoria del computer, ma finché non assemblate grossi programmi, non avrete questo problema.
Tutto ciò che vi serve è una certa esperienza ed il migliore dei modi per acquisirla, come principiante, è quello di prendere un programma già esistente, come quello riportato in queste ultime sei pagine, e quindi personalizzarlo come preferite.

ORGANIZZARE I DATI CON GLI ARRAY

Un array è un modo per memorizzare un insieme di dati e/o misure nella memoria del computer sotto forma di una tabella, in modo da poter localizzare qualsiasi elemento nella tabella senza aver bisogno di passare attraverso i precedenti. Ciascun elemento, in un array, è specificato da uno o più numeri; nel seguente array ad ogni elemento è stato assegnato una coppia di coordinate, che identifica quell'elemento e nessun altro:

	0	1	2
0	FEDERICO	CATERINA	GIOVANNI
1	100	250	840
	3	4	5
	GIANNA	ALBERTO	FRANCA
	125	223	691

Questo è un array "6x2", così chiamato perché possiede 6 colonne e 2 righe. L'elemento (1,1) è 250, l'elemento (2,0) è GIOVANNI e così via. Siccome sono necessari due numeri per identificare ciascun elemento, questo array è conosciuto come un array bidimensionale; se fosse composto da una sola riga di nomi o di numeri sarebbe necessario un singolo indice per identificare ogni termine: si tratterebbe allora di un array unidimensionale. Il comando BASIC DIM è usato per dire al computer quanto deve essere grande un array, specificando i valori massimi dell'indice per ogni dimensione dell'array.

A cosa servono gli array?
Un array unidimensionale può memorizzare una lista di numeri o di stringhe usati frequentemente:

PROGRAMMA "TABELLA UNIDIMENSIONALE"

```
LIST

10 DATA "GENNAIO","FEBBRAIO","MARZO"
20 DATA "APRILE","MAGGIO","GIUGNO"
30 DATA "LUGLIO","AGOSTO","SETTEMBRE"
40 DATA "OTTOBRE","NOVEMBRE","DICEMBRE"
50 DIM M$(11)
60 FOR N=0 TO 11
70 READ M$(N)
80 NEXT N
90 PRINT CHR$(147)
100 FOR N=0 TO 11
110 POKE 214,N+4 : PRINT : POKE 211,14
120 PRINT M$(N)
130 NEXT N
READY.
```

Qui, la riga 50 dice al computer che l'array M$ ha 12 elementi (notate che i valori dell'indice partono da 0, cosicché gli elementi dell'array sono numerati da 0 a 11). Questo programma visualizza un elenco dei mesi dell'anno, contenuti nelle righe da 10 a 40. Sebbene ci siano metodi più semplici per ottenere ciò, in un successivo programma potrete voler abbinare al mese altre informazioni, oppure il risultato di alcuni calcoli. Usando questo programma potete distinguere ogni mese specificando M$(N) dove N è il numero del mese; quando esso è in esecuzione, sarà visualizzata una immagine simile a questa, una lista dei mesi pronta per altre informazioni:

VISUALIZZAZIONE DELLA TABELLA UNIDIMENSIONALE

Come scrivere tabelle mediante gli array
Ora potete sviluppare questo programma per renderlo più utile aggiungendo, per esempio, un secondo array, un array numerico, così da poter visualizzare alcuni totali, oppure valori relativi a ciascun mese:

PROGRAMMA "TABELLA BIDIMENSIONALE"

```
LIST

10 DATA "GENNAIO","FEBBRAIO","MARZO"
20 DATA "APRILE","MAGGIO","GIUGNO"
30 DATA "LUGLIO","AGOSTO","SETTEMBRE"
40 DATA "OTTOBRE","NOVEMBRE","DICEMBRE"
50 DATA 2.5,1.0,2.5,7.5,9.5,4.5
60 DATA 3.5,4.0,4.5,4.5,4.0,2.5
70 DIM M$(11),R(11)
80 FOR N=0 TO 11
90 READ M$(N) : NEXT N
100 FOR N=0 TO 11
110 READ R(N) : NEXT N
120 PRINT CHR$(147)
130 POKE 214,3 : PRINT : POKE 211,10
140 PRINT "MESE        PIOVOSITA"
150 FOR N=0 TO 11
160 POKE 214,N+5 : PRINT
170 PRINT TAB(10);M$(N);TAB(22);R(N)
180 NEXT N
READY.
```

La tabella ha ora due intestazioni. Voi non avete bisogno di visualizzare tutti i componenti dell'array di stringhe (M$(N)) prima di giungere alla selezione dell'array numerico (R(N)): la riga 170 prende un elemento da ciascun array. Siccome gli elementi devono essere visualizzati su righe consecutive, essi possono essere facilmente identificati correlandoli al numero di riga. Per ogni valore di N, M$(N) e R(N) sono visualizzati in posizioni diverse col comando TAB) sulla riga (N+5):

VISUALIZZAZIONE DELLA TABELLA BIDIMENSIONALE

```
            MESE          PIOVOSITA

            GENNAIO          2.5
            FEBBRAIO         1
            MARZO            2.5
            APRILE           7.5
            MAGGIO           9.5
            GIUGNO           4.5
            LUGLIO           3.5
            AGOSTO           4
            SETTEMBRE        4.5
            OTTOBRE          4.5
            NOVEMBRE         4
            DICEMBRE         2.5

READY.
```

Come aggiungere un'ulteriore dimensione

Una volta capito il programma della piovosità annuale, potete ambire alla costruzione di tabelle più complesse. Nel programma di pianificazione finanziaria sottoriportato, le colonne visualizzate sono interdipendenti e voi avete la possibilità di variare alcune informazioni riportate immettendo un nuovo tasso di imposta.

Le righe 10 e 20 contengono le informazioni DATA per la prima parte della tabella: una serie di costi; la riga 30, invece, contiene di dati relativi alla seconda parte: una serie di quantità. La riga 40 contiene le coordinate che saranno usate più avanti nel programma. Le righe da 50 a 80 dimensionano un array 9x2, e memorizzano in esso le informazioni contenute nelle istruzioni DATA; le righe da 210 a 370 creano semplicemente una griglia di linee che incorniciano le informazioni. Le coordinate delle estremità inferiori delle linee verticali, memorizzate alla linea 40, sono utilizzate sotto forma di array 7x1. Le informazioni sono visualizzate nella griglia dalle righe da 120 a 200: è stata visualizzata ogni linea partendo dalla riga video 6 fino alla 14 (come impostato alla riga 130):

PROGRAMMA "TABELLA DI IMPOSTA"

```
LIST -180

10 DATA 1.98,2.40,5.60,1.05,4.35
20 DATA 2.99,1.92,7.20,5.45
30 DATA 20,19,11,45,15,15,24,4,6
40 DATA 0,8,14,17,25,32,38
50 T=15 : DIMQ(8,1),X(6)
60 FOR C=0 TO 1 : FOR R=0 TO 8
70 READ Q(R,C) : NEXT R : NEXT C
80 FOR C=0 TO 6 : READ X(C) : NEXT C
90 PRINT CHR$(147)
100 POKE 214,4 : PRINT
110 PRINT " CODICE  COSTO N.  NETTO  TAS
SA  LORDO"
120 FOR N=0 TO 8
130 POKE 214,N+6 : PRINT
140 POKE 211,2 : PRINT N+1;
150 POKE 211,8 : PRINT Q(N,0);
160 POKE 211,14 : PRINT Q(N,1);
170 POKE 211,18 : PRINT Q(N,0)*Q(N,1);
180 POKE 211,26 : PRINT (INT(((Q(N,0)*Q(
N,1)*T/100)+0.005)*100))/100;

READY.
```

PROGRAMMA "TABELLA DI IMPOSTA"

```
190 POKE 211,32 : PRINT (INT(((Q(N,0)*Q(
N,1)*(1+T/100))+0.005)*100))/100;
200 NEXT N
210 FOR C=0 TO 6 : FOR Y=4 TO 14
220 POKE 214,Y : PRINT : POKE 211,X(C)
230 PRINT CHR$(98) : NEXT Y : NEXT C
240 FOR C=0 TO 39
250 POKE 214,3 : PRINT : POKE 211,C
260 PRINT CHR$(99)
270 POKE 214,15 : PRINT : POKE 211,C
280 PRINT CHR$(99)
290 POKE 214,5 : PRINT : POKE 211,C
300 PRINT CHR$(99)
310 NEXT C : FOR C=0 TO 6
320 POKE 214,3 : PRINT : POKE 211,X(C)
330 PRINT CHR$(178)
340 POKE 214,5 : PRINT : POKE 211,X(C)
350 PRINT CHR$(123)
360 POKE 214,15 : PRINT : POKE 211,X(C)
370 PRINT CHR$(177) : NEXT C : PRINT
380 PRINT TAB(3);"PROVA CON UN'ALTRA ALI
QUOTA";
390 INPUT T : GOTO 90
READY.
```

Gli ultimi due elementi sono visualizzati dalle righe 180 e 190, che appaiono particolarmente complesse. Se il totale parziale fosse 8.25, l'imposta sarebbe stata calcolata come 0.15*8.25=1.2375, con troppe cifre decimali; per risolvere questo problema, il tasso d'imposta è moltiplicato per 100, arrotondato (INTeger) al suo valore intero (non considerando le cifre decimali), e quindi viene diviso ancora per 100. Il valore 0.005 viene aggiunto per garantire che il risultato finale sia arrotondato rispetto all'ultima cifra inserita nella tabella. Le righe da 380 a 390 invitano a immettere un nuovo tasso di imposta; se fate ciò e premete RETURN, i numeri della tabella correlati al tasso di imposta saranno ricalcolati; la possibilità di ricalcolo istantaneo è la caratteristica di un tipo di programmi per la pianificazione finanziaria chiamati "SPREADSHEET". Possono essere immessi, nelle colonne interdipendenti, valori che rappresentano i ricavi, i costi delle materie prime, i costi di produzione, le spese generali e così via. È possibile osservare direttamente gli effetti causati dalla variazione di uno o più parametri, dal momento che tutti i totali sono istantaneamente ricalcolati sul video (potete anche modificare questo tipo di programma in modo che le informazioni iniziali della tabella siano date da tastiera):

VISUALIZZAZIONE DELLA TABELLA DI IMPOSTA

CODICE	COSTO	N.	NETTO	TASSA	LORDO
1	1.98	20	39.6	5.94	45.54
2	2.4	19	45.6	6.84	52.44
3	5.6	11	61.6	9.24	70.84
4	1.05	45	47.25	7.09	54.34
5	4.35	15	65.25	9.79	75.04
6	2.99	15	44.85	6.73	51.58
7	1.92	24	46.08	6.91	52.99
8	7.2	4	28.8	4.32	33.12
9	5.45	6	32.7	4.91	37.6

PROVA CON UN'ALTRA ALIQUOTA? 12.5

RILEVARE GLI ERRORI

Sebbene progettiate meticolosamente un programma e usiate ogni cura nell'immetterlo, potreste trovare che esso si rifiuta di funzionare correttamente. In queste due pagine vedrete come effettuare il "debugging" di un programma. Avete già visto questo programma, il video gioco di pag. 45, ma questa volta contiene otto gravi errori; è come se il programma fosse stato scritto e immesso frettolosamente in modo da non funzionare. Non imbrogliate andando a vedere il programma originale!! Guardate se, controllando il programma, riuscite a trovare qualche errore, e verificate se quello che pensate errato viene corretto in queste due pagine.

Come controllare l'esecuzione di un programma
Quando eseguite un programma che contiene errori, potrete scoprirli in due modi. Primo, qualsiasi riga di programma che non ha significato per il computer produce messaggi d'errore; secondo, problemi di struttura logica o di dettaglio saranno messi in evidenza dal comportamento dell'esecuzione:

PROGRAMMA "IMPICCATO"

```
10 POKE 53280,0 : POKE 53281,0 : PRINT C
HR$(147) : PRINT TAB(14);"L'IMPICCATO"
20 POKE 214,8 : PRINT : PRINT "CHIEDI A
UN AMICO DI SCRIVERE UNA PAROLA"
30 PRINT TAB(7);"O UNA FRASE DA INDOVINA
RE"
40 PRINT : PRINT TAB(5);"** NON GUARDARE
 LO SCHERMO! **"
50 POKE 214,17 : PRINT : PRINT "   PREMI
 UN TASTO QUANDO HAI FINITO"
60 PRINT : PRINT TAB(13); : INPUT "PAROL
A = ";A$
70 PRINT "L'IMPICCATO - PREMI 1 PER IN
DOVINARE"
80 P=(32-LEN(A$))/2 : S=0 : POKE 214,7 :
 PRINT
90 FOR N=1 TO LEN(A$) : POKE 211,P
100 IF MID$(A$,N,1)=" " THEN PRINT " ";
110 PRINT "-"; : NEXT M
120 POKE 214,18 : PRINT : PRINT TAB(9);
: INPUT "PROVA UNA LETTERA";T$

READY.
```

```
LIST 130-

130 IF T$="1" THEN 180
140 S=S+1 : POKE 214,13 : PRINT : PRINT
TAB(12);"TENTATIVI =";S : POKE 214,7
150 PRINT : FOR N=1 TO LEN(A)
160 IF T$=MID$(A$,N,1) THEN POKE 211,P-N
 : PRINT T$
170 NEXT N : GOTO 120
180 POKE 214,22 : PRINT : PRINT TAB(7);
: INPUT "PROVA A INDOVINARE";T$
190 PRINT CHR$(147)
200 POKE 214,10 : PRINT : POKE 211,14
210 IF T$=A$ THEN PRINT "**ESATTO**"
220 IF T$<>A$ THEN PRINT "--SBAGLIATO--"

230 FOR T=1 TO 5 : NEXT T : GOTO 10
READY.
```

Quando proverete ad eseguire il programma, noterete che prima compare una intestazione e poi vi viene richiesto di immettere una parola; durante questa sessione di debugging immettete come stringa da indovinare "COMMODORE BASIC" in modo da ottenere gli stessi risultati mostrati qui sotto.
Dopo aver immesso queste due parole, otterrete immediatamente un messaggio d'errore:

?NEXT WITHOUT FOR ERROR IN 110

Ciò indica che il programma si è fermato perché ha incontrato qualcosa che non riesce a identificare alla riga 110. Col comando LIST 70–140 potrete vedere che alla riga 110 compare una istruzione NEXT M. Andiamo a rivedere le righe precedenti del programma, alla ricerca di una istruzione FOR, scoprirete che FOR alla riga 90 usa la variabile N e non M; come potete notare, la variabile N compare nelle righe di programma tra queste due istruzioni suggerendo che essa è quella giusta, e che M è un errore di battitura: per fare in modo che il ciclo funzioni correttamente, modificate NEXT M in NEXT N. Provate a rieseguire il programma:

VISUALIZZAZIONE DEL PROGRAMMA ERRATO

Questa volta la visualizzazione dell'intestazione e l'immissione della stringa di prova funzionano correttamente ma l'intestazione rimane sullo schermo anche quado inizia la fase seguente del gioco. Si può facilmente ovviare a questo inconveniente aggiungendo:

PRINT CHR$(147)

all'inizio della riga 70.
Un altro problema che potete osservare nella schermata precedente è che i caratteri che costituiscono le due parole da indovinare appaiono come sovrapposte. Osservando attentamente all'interno del ciclo che visualizza questi caratteri, potrete notare che la loro posizione X è determinata da una istruzione POKE alla riga 90: il valore memorizzato è costante, ed è quello della variabile P, mentre vi aspettereste che esso vari al va-

riare di N; potete rimediare cambiando il valore memorizzato alla riga 90 da P a P+N.

Come scoprire altri errori

Ora, quando eseguite il programma, non appena immettete il vostro primo tentativo di indovinare una lettera, ottenete un altro messaggio di errore:

```
?TYPE MISMATCH ERROR IN 150
```

Questo significa che il calcolatore ha incontrato un numero, o un'espressione numerica, in una posizione in cui si aspettava una stringa, o un'espressione alfanumerica, o viceversa.
Visualizzate la riga indicata dal messaggio d'errore e scoprirete che essa contiene la funzione LEN: lo scopo di questa funzione è quello di fornire la lunghezza della stringa indicata tra le parentesi; se guardate tra queste parentesi scoprirete che esse contengono la variabile A che è di tipo numerico e non di tipo stringa. Avete così trovato la causa di un altro problema; per correggerlo sostituite A con A$, che è la variabile stringa corretta, e riprovate il programma:

VISUALIZZAZIONE DEL PROGRAMMA ERRATO 2

```
L'IMPICCATO - PREMI  1  PER INDOVINARE

  D   O  --------- -----
?ILLEGAL QUANTITY  ERROR IN 160
READY.
■

              TENTATIVI = 3

         PROVA UNA LETTERA? E
```

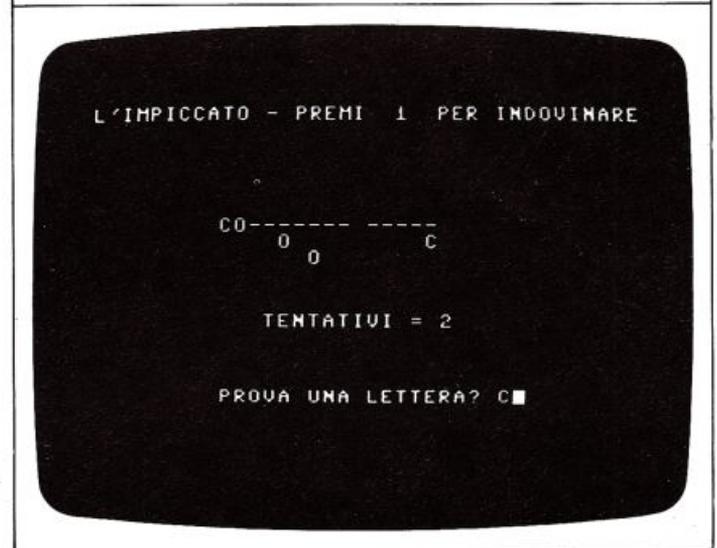
Questa volta il programma vi permette di immettere i due tentativi O e D prima di bloccarsi di nuovo, visualizzando il messaggio:

```
?ILLEGAL QUANTITY ERROR IN 160
```

Questo messaggio viene visualizzato quando un parametro ha superato i limiti dell'insieme dei valori ammessi da un certo comando. Aggiungete a ciò il fatto che lo schermo mostra le lettere T e R nella posizione e nell'ordine sbagliati e avrete una buona ragione per sospettare che la causa del problema sia l'istruzione POKE alla riga 160.
Per confermare questo sospetto potete visualizzare il valore che è stato memorizzato:

```
PRINT P-N
```

Otterrete un risultato negativo che non è certamente il valore corretto da memorizzare; siccome le lettere sono visualizzate nell'ordine sbagliato, il valore di N deve essere sommato e non sottratto al valore di P:

VISUALIZZAZIONE DEL PROGRAMMA ERRATO 3

Ora potete vedere che quando immettete T e R come tentativi, le lettere sono visualizzate sullo schermo nelle corrette posizioni orizzontali e nel giusto ordine. Sfortunatamente non avete scoperto un altro errore: le lettere che compaiono più di una volta nella stringa non sono visualizzate sulla stessa linea; è come se fosse stato premuto il tasto RETURN dopo la visualizzazione di ogni lettera.
Questo errore può essere facilmente individuato nell'istruzione PRINT alla fine della riga 160, la quale dovrebbe terminare con un punto e virgola per sopprimere RETURN.
Se eseguite di nuovo il programma dopo aver corretto questi errori, esso produrrà degli ottimi risultati finché introducete le lettere T, R, A, e C ma, quando battete la lettera I, esso smette di chiedervi di indovinare un'altra lettera e vi invita a tentare di indovinare l'intera frase. Secondo le istruzioni del gioco, ciò può avvenire solo se voi immettete il numero 1 invece di una lettera; ma il numero 1 e la lettera I si assomigliano molto perciò, se controllate la riga 130, scoprirete che chi ha immesso il programma si è confuso: un errore facile da correggere.
L'ultimo errore è banale. Quando voi avete indovinato alcune lettere e tentate quindi di indovinare l'intera parola, il programma dovrebbe dirvi se il tentativo ha avuto successo, ma l'ultimo errore non permette al messaggio relativo di rimanere sullo schermo abbastanza a lungo per essere letto. Per eliminare questo ultimo problema aumentate il ciclo di ritardo alla riga 230 da 5 a 5000. Ora potete confrontare questo programma con quello di pag. 45 per verificare che i due programmi funzionino in modo identico.

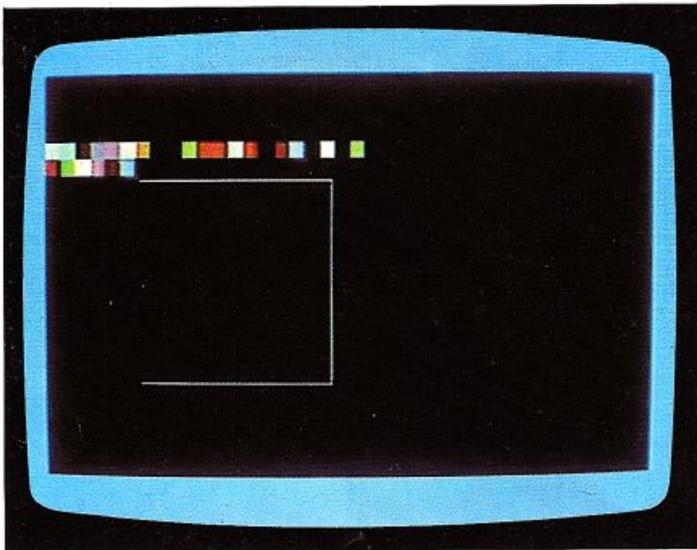
Come evitare di commettere errori nella programmazione

Mentre sviluppate i vostri programmi, un costante controllo vi consentirà di eliminare la maggior parte degli errori dalla versione finale. Quando state controllando un programma che avete scritto, cercate di immaginare tutte le situazioni che si potrebbero verificare durante il suo funzionamento, facendo particolare attenzione ad ogni limite numerico. Se avete previsto delle istruzioni che prevengano l'arresto del programma in alcune circostanze, controllate attentamente anche queste.

CONSIGLI E SUGGERIMENTI

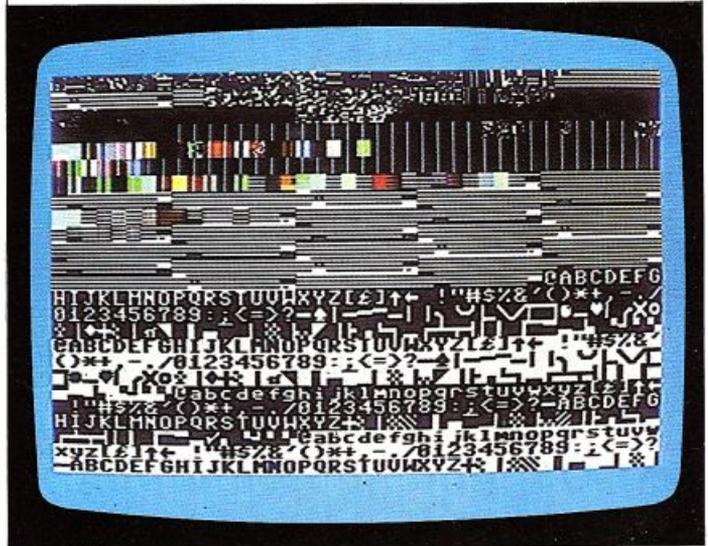
Uno dei più grossi problemi che avete incontrato durante la lettura di questo libro è l'immissione dei programmi: è molto difficile ottenere un programma corretto, la prima volta: normalmente esso presenta messaggi di errore. Quando state usando l'alta risoluzione però, i messaggi d'errore non possono essere letti; il vostro programma si arresta e tutto quello che potete vedere del messaggio d'errore è una riga di quadratini colorati. Cosa potete fare per trovare i vostri errori?

Come identificare gli errori in alta risoluzione
La seguente figura mostra un programma grafico che usa le subroutine per alta risoluzione e ciò che esso produce sullo schermo. Ovviamente c'è qualcosa che non va:

PROGRAMMA ERRATO IN ALTA RISOLUZIONE

Dal momento che l'area di memoria che normalmente contiene il testo ora contiene la memoria colore, non potete visualizzare lettere sullo schermo in alta risoluzione.
Normalmente, per uscire dall'alta risoluzione vengono usati i comandi RUN/STOP e RESET; tuttavia, questi comandi, cancellano lo schermo senza lasciarvi alcuna spiegazione circa l'arresto del programma. Quello di cui avete bisogno è un metodo per passare dall'alta alla bassa risoluzione senza cancellare lo schermo; potete ottenere ciò immettendo i seguenti comandi subito dopo l'emissione del messaggio d'errore. Innanzitutto immettere questa riga:

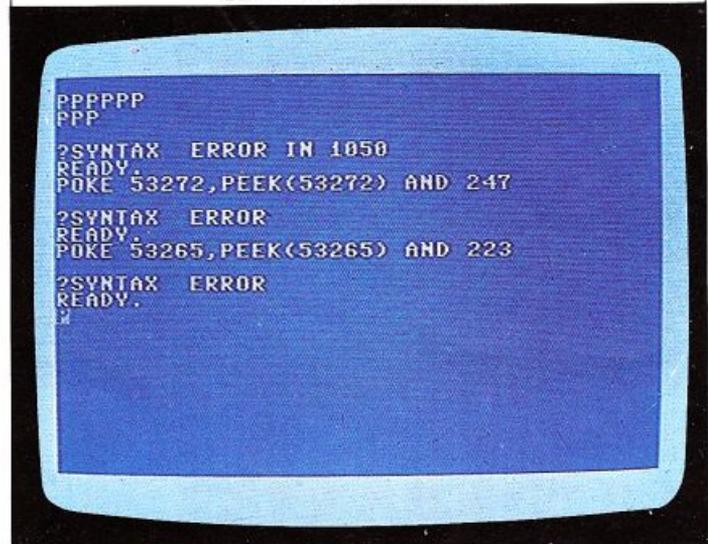
```
POKE 53272,PEEK(53272) AND 247
```

Sullo schermo apparirà immediatamente un'immagine simile a questa:

ABBANDONANDO L'ALTA RISOLUZIONE

Ora immettete una seconda riga di comandi per visualizzare i messaggi d'errore sullo schermo:

```
POKE 53265,PEEK(53265) AND 223
```

Questa riga mostra cosa dovreste vedere ora:

GLI ERRORI RIVELATI

Come potete osservare questo metodo è efficace, sebbene ci sia un problema: i due comandi POKE sono piuttosto complessi, e siete costretti ad immetterli alla cieca, dal momento che non potete vedere sullo schermo quello che state immettendo.
Una migliore soluzione è di accettare il fatto che probabilmente farete errori durante lo sviluppo del programma, e di prepararsi a questa evenienza. Quello che dovete fare è immettere nelle vostre subroutine grafiche le precedenti righe, aggiungendo l'istruzione END come è stato fatto qui:

```
20 POKE 53272,PEEK(53272) AND 247
30 POKE 53265,PEEK(53265) AND 223
40 END
```

Adesso, se il vostro programma in alta risoluzione si arresta, tutto quello che dovete fare è di immettere il comando GOTO 20 e appariranno i messaggi d'errore.

Come utilizzare meglio il comando RESTORE
Una opzione assente dal repertorio del BASIC del Commodore è la possibilità di impostare il puntatore del computer ad una qualsiasi istruzione DATA contenuta nel vostro programma. Per esempio non potete immettere:

```
10 RESTORE 50
```

che significa: "Ripristina il DATA pointer all'inizio della riga 50 invece che all'inizio della prima istruzione DATA nel programma". Questa possibilità può rivelarsi molto utile quando devono essere memorizzati molti messaggi di testo, come per esempio, in un programma di estratti conto o in un video gioco, ed è necessario accedere ad essi molto velocemente. Il grosso vantaggio è che non è necessario occupare altro spazio in memoria oltre alle righe DATA nel programma, e non è richiesto il dimensionamento di un array per la lettura delle informazioni, dal momento che esse possono essere lette direttamente dalle istruzioni DATA. La seguente figura mostra una subroutine, che parte dalla riga 5000, la quale carica una piccola routine in codice macchina in un'area di memoria del computer non utilizzata:

ROUTINE DI RESTORE IN CODICE MACCHINA

```
LIST

5000 FOR C=0 TO 23
5010 READ A
5020 POKE 49494+C,A
5030 NEXT C
5040 DATA 165,63,133,20,165,64
5050 DATA 133,21,32,19,166,165
5060 DATA 95,233,1,133,65,165
5070 DATA 96,233,0,133,66,96
READY.
```

Questo blocco deve essere riportato una sola volta nel programma; esso contiene le istruzioni necessarie al computer per cambiare il DATA pointer con il valore della riga di DATA usata attualmente.
La prossima figura mostra una subroutine molto piccola che usa questo blocco in codice macchina per produrre lo stesso effetto di RESTORE ad un certo numero di riga:

COME USARE LA ROUTINE DI RESTORE

L'Entry Point per questa subroutine è alla riga 5500; potete impostare il valore della variabile RN al numero di riga che volete assegnare al DATA pointer e quindi immettere l'istruzione GOSUB 5500.

Dove memorizzare le subroutine in codice macchina
Come avete appena visto, è spesso vantaggioso ricorrere a brevi routine in codice macchina all'interno di un programma BASIC. Normalmente questo viene fatto usando il comando BASIC SYS che è seguito da un numero che rappresenta l'indirizzo della memoria da cui parte la routine in codice macchina.
Il comando SYS è molto simile al comando BASIC GOSUB, in quanto, dopo aver eseguito la routine in codice macchina, il controllo ritorna all'istruzione che segue al comando SYS.
Per essere sicuri che il programma ritorni nel punto giusto, dovete immettere un comando in codice macchina equivalente a RETURN che è RTS (ReTurn from Subroutine); la rappresentazione decimale di questo comando è 96, valore che potete notare alla fine delle righe DATA del programma.
Quando volete utilizzare delle subroutine in codice macchina nel vostro programma in BASIC, uno dei problemi che potreste incontrare è quello di decidere in quale locazione di memoria caricare i byte che compongono il codice macchina. Nel Commodore questo problema è facile da risolvere: c'è un'area della RAM dall'indirizzo 49152 all'indirizzo 53247 (4K in tutto), che è a disposizione delle routine in codice macchina e non è utilizzata da altre funzioni del computer. Questa è l'area ideale per la memorizzazione delle routines in codice macchina. La routine di RESTORE nella figura a sinistra è caricata appunto in questa zona "protetta" di memoria.

GRIGLIE PER L'ALTA RISOLUZIONE

Le due griglie seguenti sono utili per permettervi di identificare la locazione di memoria video relativa a un pixel o a un gruppo di pixel. La prima è corredata da due insiemi di valori per ogni dimensione: i valori più interni rappresentano semplicemente le coordinate orizzontali e verticali; quelli più all'esterno indicano invece le locazioni di memoria relative ad ogni quadratino della griglia.
I pixel di ogni quadratino sono controllati da otto locazioni di memoria consecutive; per individuare l'indirizzo della locazione più bassa della serie di otto, è sufficiente sommare i due numeri riportati più all'esterno in corrispondenza del quadratino che vi interessa. Potete ora spostarvi alla griglia 8x8 qui sotto.

Come controllare i singoli pixel

Una volta che avete stabilito quali sono le otto locazioni che corrispondono al quadratino che vi interessa, dovete memorizzarvi i valori necessari per predisporre i singoli pixel. Ogni locazione di memoria corrisponde ad una riga di pixel così, a partire dall'alto verso il basso, vengono interessate proprio otto locazioni di memoria consecutive. Nella griglia sulla destra sono stati accesi sei pixel sulla riga 2; per compiere questa operazione è necessario, innanzi tutto, ricavare la locazione di inizio del quadratino e, quindi, sommarvi 2 per ottenere l'indirizzo esatto del byte corrispondente alla riga 2. Infine dovrete sommare i valori decimali dei pixel accesi (128+64+32+8+2+1 in questo esempio) e memorizzare il risultato in quella locazione. Se voleste accendere dei pixel che si trovano su più di una riga, dovreste eseguire l'operazione precedente per più di una locazione di memoria.
Lavorare in questo modo con la memoria video per l'alta risoluzione è molto utile per comprendere esattamente come il Commodore opera in grafica ad alta risoluzione.
Provate, e vi farete un'idea esatta del funzionamento delle subroutine grafiche presentate all'inizio di questo volume.

GRIGLIA PER GLI SPRITE

Gli sprite del Commodore sono composti da 21 righe di 24 pixel ciascuna; ogni pixel è controllato da un singolo bit mediante un'istruzione di POKE.
Potete utilizzare la griglia seguente per disegnare i vostri sprite e calcolare i valori da memorizzare, che possono essere immessi in un'istruzione DATA del vostro programma.
Dopo aver disegnato (a matita) il vostro sprite sulla griglia, sommate i valori decimali di tutti i pixel accesi in ogni riga e riportate il totale sulla colonna di destra.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
	128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1
0																								
1																								
2																								
3																								
4																								
5																								
6																								
7																								
8																								
9																								
10																								
11																								
12																								
13																								
14																								
15																								
16																								
17																								
18																								
19																								
20																								

valori per DATA		

Posizionare gli sprite
Le coordinate della posizione di uno sprite possono assumere un qualsiasi valore da 0 a 225 (in verticale) e da 0 a 511 (in orizzontale), tuttavia solo una parte di questi valori corrisponde a posizioni realmente visibili sullo schermo. Lo schema sulla destra mostra tutte queste coordinate in relazione allo schermo: il riquadro centrale rappresenta l'area visibile sullo schermo, che va da 24 a 343 (in orizzontale) e da 30 a 229 (in verticale). Tutto ciò significa che potete facilmente muovere gli sprite fuori e dentro lo schermo.
Dal momento che la posizione verticale varia da 0 a 255, essa può essere controllata da un singolo byte; la posizione orizzontale, d'altra parte, ha bisogno di due byte se volete utilizzare tutti i valori da 0 a 511 (=2x256).
Se volete che uno sprite continui a muoversi oltre la posizione orizzontale 255, dovete impostare di conseguenza la locazione V+16.

CODICI MEMORIA VIDEO

In bassa risoluzione è possibile accedere ad ogni punto dello schermo del Commodore usando semplicemente l'istruzione POKE; ciò vi permette di visualizzare un carattere in un colore qualsiasi in una qualsiasi posizione dello schermo.

Sono necessari, a questo scopo, due istruzioni POKE: una per specificare il carattere da visualizzare in quella posizione, e l'altra per definirne il colore. La griglia seguente vi aiuta nell'individuare le giuste locazioni della memoria video e della memoria colore.

Memoria video e memoria colore
Per individuare le due locazioni di memoria corrispondenti a una posizione dello schermo, dovete, per prima cosa, determinare le locazioni di memoria relative alla posizione verticale; potete quindi, determinare le locazioni delle memorie video e colore sommando a tali valori il numero di colonna. Determinate queste locazioni, potete memorizzarvi i codici relativi a un carattere e al suo colore.

Memoria video	Memoria colore
1024	55296
1064	55336
1104	55376
1144	55416
1184	55456
1224	55496
1264	55536
1304	55576
1344	55616
1384	55656
1424	55696
1464	55736
1504	55776
1544	55816
1584	55856
1624	55896
1664	55936
1704	55976
1744	56016
1784	56056
1824	56096
1864	56136
1904	56176
1944	56216
1984	56256

I codici dei caratteri
La tabella seguente riporta i codici relativi alle lettere maiuscole e minuscole della tastiera del Commodore: questi sono i codici che devono essere memorizzati nella memoria video. I codici da 128 a 255 producono un'immagine in reverse dei caratteri mostrati qui sotto.

	0	1	2	3	4	5	6	7	8	9
0	@	A a	B b	C c	D d	E e	F f	G g	H h	I i
10	J j	K k	L l	M m	N n	O o	P p	Q q	R r	S s
20	T t	U u	V v	W w	X x	Y y	Z z	[	£	]
30	↑	←	SPAZIO	!	"	#	$	%	&	'
40	(	)	*	+	,	–	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?		♠ A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	● Q	R	♥ S	T	U	V	○ W	♣ X	Y
90	♦ Z				π		SPAZIO			
100										
110										
120			✓							

L'INSIEME DEI CODICI ASCII

L'insieme dei codici ASCII costituisce una sequenza di caratteri e funzioni a cui si può accedere mediante il comando CHR$; si tratta di un codice digitale standard per i caratteri usati dai computer: i codici da 33 a 127 rappresentano gli stessi caratteri su quasi tutti i calcolatori, mentre gli altri codici sono usati in modo diverso da ciascuna macchina. Sul Commodore tali codici controllano un certo insieme di funzioni, quali la predisposizione dei colori e la rappresentazione dei caratteri speciali. Il codice ASCII usa solo 7 bit di ogni byte, lasciando così spazio per un bit di "partita" usato per il rilevamento di errori di trasmissione in linea.

	0	1	2	3	4	5	6	7	8	9
0						Bianco			Disabilità **SHIFT+C=**	Abilità **SHIFT+C=**
10				**RETURN**	Lettere maiuscole			Cursore in giù	**RVS ON**	**CLR HOME**
20	**INST/DEL**								Rosso	Cursore a destra
30	Verde	Blu	**SPAZIO**	!	"	#	$	%	&	'
40	(	)	*	+	,	–	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	£	]	↑	←		♠		
100										
110				●		♥			╳	○
120	♣		♦				π			Arancio
130				Tasto Funzione 1	Tasto Funzione 2	Tasto Funzione 3	Tasto Funzione 4	Tasto Funzione 5	Tasto Funzione 6	Tasto Funzione 7
140	Tasto Funzione 8	**SHIFT + RETURN**	Lettere maiuscole		Nero	Cursore in su	**RVS OFF**	**CLR HOME**	**INST/DEL**	Marrone
150	Rosa	Grigio chiaro	Grigio	Verde chiaro	Azzurro	Grigio scuro	Porpora	Corsore a sinistra	Giallo	Ciano
160	**SPAZIO**									
170										
180										
190				♠						
200										●
210		♥			╳	○	♣		♦	
220			π		**SPAZIO**					
230										
240										
250						π				

GLOSSARIO

I termini in **grassetto** sono parole chiave del BASIC.

ABS
Fornisce il valore assoluto di un numero.

AND
Opera su due condizioni o due valori binari fornendo come risultato "1" (VERO) solo se entrambe le condizioni sono vere o entrambi i valori sono a livello logico "1".

ASC
Fornisce il codice ASCII di un carattere.

ASCII
American Standard Code for Information Interchange; è il sistema di codifica dei caratteri usato dal Commodore.

BASIC
Beginner's All-purpose Symbolic Instruction Code; è il linguaggio di programmazione ad alto livello più comunemente usato.

Binario
È il sistema di numerazione usato dai computer e basato solo su due cifre: "0" e "1".

Bit
È la cifra binaria: "0" o "1".

Byte
È un insieme di otto bit.

Chip
È un singolo componente contenente un intero circuito elettronico. È chiamato anche "circuito integrato" (IC = integrated circuit).

CHR$
Converte un codice ASCII nel carattere corrispondente.

COS
Fornisce il coseno di un angolo.

CPU
Central Processing Unit. È costituita in generale da un solo chip detto "microprocessore", il quale esegue le operazioni aritmetiche e gestisce il resto del computer.

Cursore
È un simbolo lampeggiante che appare sullo schermo per indicare dove apparirà il prossimo carattere.

DATA
Il programma tratta tutti i valori che seguono la parola chiave **DATA** come informazioni che saranno necessarie nel corso del programma stesso. È usato in coppia con **READ**.

Debugging
È l'operazione di eliminazione degli errori (bug) da un programma.

DEF FN
Definisce una funzione.

DIM
Informa il computer della dimensione di un array.

END
Termina il programma (vedere anche **STOP**).

Filename (nome di file)
È il nome assegnato ad un programma o ad un insieme di dati che è possibile memorizzare su nastro o su disco.

FN
Indica che la variabile che segue rappresenta una funzione (vedere anche **DEF FN**).

FOR...NEXT
È un ciclo che ripete una sequenza di istruzioni del programma per un numero determinato di volte.

GOSUB
Fa saltare il programma alla subroutine che comincia al numero di riga che segue questa parola chiave. La subroutine deve sempre terminare con **RETURN**.

GOTO
Fa saltare il programma al numero di riga specificato dopo questa parola chiave.

IF...THEN
Forza il computer a compiere un certo insieme di operazioni se la condizione specificata è vera.

INPUT
Quando è usato in un programma, permette al computer di aspettare che dei dati siano immessi da tastiera.

INT
Arrotonda un numero alla sua parte intera.

Inviluppo
È la variazione dell'intensità di una nota durante la sua esecuzione. È possibile selezionare diversi inviluppo mediante un'istruzione **POKE**.

K
È l'abbreviazione di Kilobyte (= 1024 byte).

LEFT$
Costruisce una stringa utilizzando i caratteri più a sinistra di un'altra stringa.

LEN
Fornisce il numero di caratteri da cui è composta una stringa.

LET
Assegna un valore ad una variabile. L'uso di **LET** è opzionale sul Commodore.

LIST
Permette di visualizzare sullo schermo il programma attualmente memorizzato.

LOAD
Trasferisce un programma dal nastro (o dal disco) alla memoria del computer.

Loop
Si tratta di una sequenza di istruzioni del programma che vengono ripetutamente eseguite fino a che non si verifica una specifica condizione.

MID$
Costruisce una stringa utilizzando i caratteri di mezzo di un'altra stringa.

NEW
Cancella il programma attualmente in memoria.

ON...GOTO/GOSUB
Fa saltare il programma a uno dei numeri di riga specificato di seguito, a seconda del valore assunto da una variabile di controllo.

OR
Opera su due condizioni o due valori binari formando come risultato "1" (vero) se almeno una delle due condizioni è vera o uno dei due valori è a livello logico "1".

PEEK
Legge il valore memorizzato in una specifica locazione di memoria.

POKE
Memorizza un valore in una specifica locazione di memoria.

PRINT
Permette di visualizzare sullo schermo i termini che seguono questa parola chiave.

READ
Permette al calcolatore di accedere alle informazioni contenute nelle istruzioni **DATA**.

REM
Permette di inserire commenti in un programma. Il computer ignorerà tutto ciò che segue la parola chiave **REM** in una riga di programma.

RESTORE
Predispone il computer per la lettura del primo termine di un insieme di istruzioni **DATA**.

RETURN
Termina una subroutine (vedere anche **GOSUB**).

RIGHT$
Costruisce una stringa utilizzando i caratteri più a destra di un'altra stringa.

RND
Fornisce un numero casuale all'interno di un intervallo specificato.

SAVE
Salva il programma in memoria su disco o su nastro. Il programma è identificato da un nome.

SGN
Fornisce il segno di un numero.

SID
Sound Interface Device. È il chip usato dal Commodore per produrre i suoni.

SIN
Fornisce il seno di un angolo.

Sprite
È un simbolo grafico, definibile mediante delle istruzioni **POKE**, che si può muovere sullo schermo.

SQR
Fornisce la radice quadrata di un numero.

STEP
Imposta il passo di un ciclo **FOR...NEXT**.

STOP
Termina il programma e visualizza il numero di riga a cui si trova.

Stringa
È una sequenza di caratteri che vengono trattati come un singolo termine: il nome di qualcuno, per esempio.

Subroutine
È una parte del programma che può essere richiamata quando è necessario, per esempio allo scopo di costruire una particolare immagine o di eseguire ripetutamente una certa sequenza di calcoli.

SYS
Lancia l'esecuzione di una routine in codice macchina.

TAB
Specifica la posizione in cui verrà stampato il testo su una riga video.

Variabile
È un insieme di locazioni di memoria a cui è assegnato un nome e in cui, durante lo svolgimento del programma, possono essere memorizzate delle informazioni.

VERIFY
Controlla che un programma sia stato salvato correttamente su nastro o su disco per mezzo di **SAVE**.

VIC
Video Interface Circuit. È il chip che si occupa del controllo dello schermo.

INDICE ANALITICO

Le principali occorrenze sono riportate in **grassetto**

Ringraziamenti
La Dorling Kindersley ringrazia Ian Graham per il suo fondamentale contributo a questa serie. Un ringraziamento anche a Philip Freebrey e a Paul Rubert per l'assistenza tecnica, a Fred Gill per aver ricontrollato il testo e a Richard Bird per l'indice analitico.
La Commodore Business Machine (UK) Ltd ha gentilmente fornito le apparecchiature.

ScreenShot

COLLANA DI PROGRAMMAZIONE

Un corso originale e divertente, di concezione completamente nuova, per imparare da soli a programmare il Commodore 64. Più di 150 fotografie originali di listati di programmi e di programmi in azione, per mostrare sulla pagina esattamente ciò che accade sullo schermo video. Pieno di suggerimenti sulle tecniche di programmazione, di schemi e tabelle di riferimento, di consigli su come ottenere i risultati migliori dal vostro Commodore 64.

ALCUNI ARGOMENTI

Programmazione in grafica ad alta risoluzione • Curve e cerchi • Grafica 'naturale' • Come progettare da sé i caratteri • Animazione di sprite • Sovrapposizioni e collisioni di sprite • Diagrammi a torta • Grafici e diagrammi a barre • Come programmare i video-games • I suoni col computer

GIA' PUBBLICATI NELLA COLLANA DI PROGRAMMAZIONE *ScreenShot*

COME PROGRAMMARE PASSO PER PASSO

COMMODORE 64 *LIBRO 1*

ZX SPECTRUM *LIBRO 1 - 2*

0026442–4

(18137) Lire 18.500